

Oracle Pushdown Automata, Nondeterministic Reducibilities, and the Hierarchy over the Family of Context-Free Languages

TOMOYUKI YAMAKAMI*

Abstract: We impose various oracle mechanisms on nondeterministic pushdown automata, which naturally induce nondeterministic reducibilities among formal languages in a theory of context-free languages. In particular, we examine a notion of nondeterministic many-one CFL reducibility and conduct a ground work to formulate a coherent framework for further expositions. Two more powerful reducibilities—bounded truth-table and Turing CFL-reducibilities—are also discussed in comparison. The Turing CFL-reducibility, in particular, makes it possible to induce a useful hierarchy built over the family CFL of context-free languages. Basic structural properties are proven for each level of this CFL hierarchy. The first and second levels of the hierarchy are proven to be different. The rest of the hierarchy (more strongly, the Boolean hierarchy built over each level of the CFL hierarchy) is also infinite unless the polynomial hierarchy over NP collapses. This follows from a characterization of the Boolean hierarchy over the k th level of the polynomial hierarchy in terms of the Boolean hierarchy over the $k + 1$ st level of the CFL hierarchy. Similarly, the complexity class Θ_k^P is related to the k th level of the CFL hierarchy. We argue that the CFL hierarchy coincides with a hierarchy over CFL built by application of many-one CFL-reductions. We show that BPCFL—a bounded-error probabilistic version of CFL—is not included in CFL even in the presence of advice. Moreover, we exhibit a relativized world where BPCFL is not located within the second level of the CFL hierarchy.

Keywords: regular language, context-free language, pushdown automaton, oracle, many-one reducibility, Turing reducibility, truth-table reducibility, CFL hierarchy, polynomial hierarchy, advice, Dyck language

1 Backgrounds and Main Themes

A fundamental notion of *reducibility* has long played an essential role in the development of a theory of NP-completeness. In the 1970s, various forms of polynomial-time reducibility emerged mostly based on a model of oracle Turing machine and they gave a means to study *relativizations* of associated families of languages. Most typical reducibilities in use today in computational complexity theory include many-one, truth-table, and Turing reducibilities obtained by imposing appropriate restrictions on the functionality of oracle mechanism of underlying Turing machines. Away from standard complexity-theoretical subjects, we will shift our attention to a theory of formal languages and automata. Within this theory, we wish to lay out a framework for a future extensive study on structural complexity issues by providing a solid foundation for various notions of reducibility and their associated relativizations.

Of many languages, we are particularly interested in *context-free languages*, which are characterized by context-free grammars or one-way nondeterministic pushdown automata (or npda's, hereafter). The context-free languages are inherently nondeterministic. In light of the fact that the notion of nondeterminism appears naturally in real life, it has become a key to many fields of computer science. The family CFL of context-free languages has proven to be a fascinating subject, simply because every language in CFL behaves quite differently from the corresponding nondeterministic polynomial-time class NP. For instance, whereas NP is closed under any Boolean operations except for complementation, CFL is not even closed under intersection. This non-closure property is caused by the lack of flexibility in the use of memory storage by an underlying model of npda. On the contrary, a restricted use of memory helps us prove a separation between the first and the second levels of the Boolean hierarchy $\{CFL_k \mid k \geq 1\}$ built over CFL, which was defined in [24], by applying Boolean operations (intersection and union) alternately to CFL. Moreover, we can prove that a family of languages $CFL(k)$ composed of intersections of k context-free languages truly forms an infinite hierarchy [12]. Such an architectural restriction sometimes becomes a crucial issue in certain applications of pushdown automata. For instance, a one-way probabilistic pushdown automaton (or ppda) with bounded-error probability in general cannot amplify its success probability [8].

A most simple type of reduction is probably *many-one reduction* and, by adopting the existing formulation of this reducibility, we intend to bring a notion of nondeterministic many-one reducibility into context-free languages under the name of *many-one CFL-reducibility*. We write CFL_m^A to denote the family of languages

*Present Affiliation: Department of Information Science, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan

that are many-one CFL-reducible to a given oracle A . Notice that Reinhardt [14] earlier considered many-one reductions that are induced by nondeterministic finite automata (or nfa's), which use no memory space. Our goal is to build a hierarchy of language families over CFL using our reducibility by way of immediate analogy with constructing the *polynomial(-time) hierarchy* over NP [15, 16]. For this purpose, we choose npda's rather than nfa's. Owing mostly to the unique architecture of npda's, our reducibility exhibits quite distinctive features; for instance, this reducibility in general does not admit a transitivity property. (For this reason, our reducibility might have been called a “quasi-reducibility” if the transitive property is a prerequisite for a reducibility notion.) As a consequence, the family CFL is not closed under the many-one CFL-reducibility (that is, $\text{CFL}_m^{\text{CFL}} \neq \text{CFL}$). This non-closure property allures us to study the family $\text{CFL}_{m[k]}^{\text{CFL}}$ whose elements are obtained by the k -fold application of many-one CFL-reductions to languages in CFL. As shown in Section 3.1, the language family $\text{CFL}_{m[k]}^{\text{CFL}}$ turns out to coincide with $\text{CFL}_m^{\text{CFL}(k)}$.

We further discuss two more powerful reducibilities in use—bounded truth-table and Turing CFL-reducibilities based on npda's. In particular, the Turing CFL-reducibility introduces a hierarchy analogous to the polynomial hierarchy: the hierarchy $\{\Delta_k^{\text{CFL}}, \Sigma_k^{\text{CFL}}, \Pi_k^{\text{CFL}} \mid k \geq 0\}$ built over CFL, which we succinctly call the *CFL hierarchy*, and this hierarchy turns out to be quite useful in classifying the computational complexity of a certain group of languages. As a quick example, the languages $\text{Dup}_2 = \{xx \mid x \in \{0, 1\}^*\}$ and $\text{Dup}_3 = \{xxx \mid x \in \{0, 1\}^*\}$, which are known to be outside of CFL, fall into the second level Σ_2^{CFL} of the CFL hierarchy. A simple matching language $\text{Match} = \{x\#w \mid \exists u, v [w = uv]\}$ is in Σ_2^{CFL} . Two more languages $\text{Sq} = \{0^n 1^{n^2} \mid n \geq 1\}$ and $\text{Prim} = \{0^n \mid n \text{ is a prime number}\}$ belong to Σ_3^{CFL} and Π_3^{CFL} , respectively. A slightly more complex language $\text{MulPrim} = \{0^{mn} \mid m \text{ and } n \text{ are prime numbers}\}$ is also in Σ_3^{CFL} . The first and second levels of the CFL hierarchy are proven to be different; more strongly, we can prove that $\Sigma_2^{\text{CFL}} \not\subseteq \Sigma_1^{\text{CFL}}/n$, where Σ_1^{CFL}/n is a non-uniform version of Σ_1^{CFL} , defined in [17] and further explored in [22]. Regarding the aforementioned language families $\text{CFL}(k)$ and CFL_k , we can show later that the families $\text{CFL}(\omega) = \bigcup_{k \geq 1} \text{CFL}(k)$ and $\text{BHCFL} = \bigcup_{k \geq 1} \text{CFL}_k$ belong to the second level $\Sigma_2^{\text{CFL}} \cap \Pi_2^{\text{CFL}}$ of the CFL hierarchy, from a fact that $\text{CFL}(\omega) \subseteq \text{BHCFL}$. Despite obvious similarities between their definitions, the CFL hierarchy and the polynomial hierarchy are quite different in nature. In Section 4.1, we show that $\text{CFL}_m^{\text{CFL}(\omega)}$ is located within Σ_3^{CFL} . Because of npda's architectural restrictions, “standard” techniques of simulating a two-way Turing machine, in general, do not apply; hence, we need to develop new simulation techniques for npda's.

In this paper, we employ three simulation techniques to obtain some of the aforementioned results. The first technique is of guessing and verifying a *stack history* to eliminate a use of stack, where a stack history means a series of consecutive stack operations made by an underlying npda. The second technique is applied to the case of simulating two or more tape heads by a single tape head. To adjust the different head speeds, we intentionally insert extra dummy symbols to generate a single query word so that an oracle can eliminate them when it accesses the query word. The last technique is to generate a string that encodes a computation path generated by a nondeterministic machine. All the techniques are explained in details in Sections 3.1–3.2. Those simulation techniques actually make it possible to obtain three alternative characterizations of the CFL hierarchy in Section 4.2.

Reinhardt [14] related the aforementioned hierarchy of his to another hierarchy defined by alternating pushdown automata and he gave a characterization of the polynomial hierarchy by this alternating hierarchy using logarithmic-space (or log-space) many-one reductions. Using an argument similar to his, we can establish in Section 5 an exact characterization of the e th level of the Boolean hierarchy over the k th level Σ_k^{P} of the polynomial hierarchy in terms of the corresponding e th level of the Boolean hierarchy over the $k + 1$ st level of the CFL hierarchy. Moreover, we give a new characterization of Θ_k^{P} (i.e., Wagner's [19] notation for $\text{P}_T(\Sigma_{k-1}^{\text{P}}[O(\log n)])$) in terms of the k th level of the CFL hierarchy using log-space truth-table reductions. As an immediate consequence, all levels of the Boolean hierarchy over each level of the CFL hierarchy are different unless the polynomial hierarchy collapses.

Another relevant notion induced by reducibility is a *relativization* of language families. For issues not settled by the current knowledge of us, we often resort to a relativization, which helps us discuss the existence of various relativized worlds in which a certain relationship among target language families either holds or fails. For instance, we can construct in Section 4.3 a recursive oracle for which the family BPCFL of languages recognized by bounded-error one-way ppda's is not included within the second level of the CFL hierarchy. (Of course, there also exists an obvious oracle that makes this inclusion hold.) This separation result contrasts a well-known fact that BPP is included in $\Sigma_2^{\text{P}} \cap \Pi_2^{\text{P}}$ in any relativized world. To deal with oracle-dependent languages in a relativized CFL hierarchy, we utilize its characterization by bounded-depth Boolean circuits of alternating ORs and ANDs. Our proof relies on a special form of the well-known *switching lemma* [1],

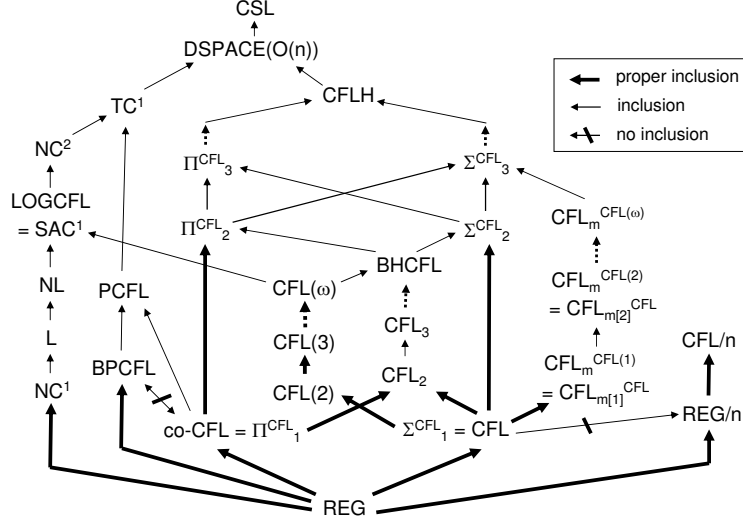


Figure 1: Hasse diagram of inclusion relations among language families

in which a circuit of OR of ANDs can be transformed into another equivalent circuit of AND of ORs by partially setting 0 and 1 to input variables. In the unrelativized world, we can prove that $\text{BPCFL} \not\subseteq \text{CFL}/n$. This separation extends a known result of [8] that $\text{BPCFL} \not\subseteq \text{CFL}$.

A Hasse diagram in Fig.1 summarizes some of the inclusion relationships among language families discussed so far. The notation CFLH in the figure denotes the union $\bigcup_{k \geq 1} (\Sigma_k^{\text{CFL}} \cup \Pi_k^{\text{CFL}})$.

Although most results in this paper are embryonic, we strongly believe that these results will pave a road to more exciting discoveries in *structural complexity theory of formal languages and automata*.

2 A Preparation for Our Expositions

We will briefly explain basic notions and notations that help the reader go through the subsequent sections. Generally, we will follow the existing terminology in a field of formal languages and automata. However, the reader who is familiar with computational complexity theory needs extra attentions to certain notations (for instance, $\text{CFL}(k)$ and CFL_k) that are used in quite different ways.

2.1 Alphabets, Strings, and Languages

Given a finite set A , the notation $\|A\|$ expresses the number of elements in A . Let \mathbb{N} be the set of all *natural numbers* (i.e., nonnegative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. For any number $n \in \mathbb{N}^+$, $[n]$ denotes the integer set $\{1, 2, \dots, n\}$. The term “polynomial” always means a polynomial on \mathbb{N} with coefficients of non-negative integers. In particular, a *linear polynomial* is of the form $ax + b$ with $a, b \in \mathbb{N}$. The notation $A - B$ for two sets A and B indicates the *difference* $\{x \mid x \in A, x \notin B\}$ and $\mathcal{P}(A)$ denotes the *power set* of A ; that is, the collection of all subsets of A .

An *alphabet* is a nonempty finite set Σ and its elements are called *symbols*. A *string* x over Σ is a finite series of symbols chosen from Σ and its *length*, denoted $|x|$, is the total number of symbols in x . The *empty string* λ is a special string whose length is zero. Given a string $x = x_1x_2 \cdots x_{n-1}x_n$ with $x_i \in \Sigma$, x^R represents the *reverse* of x , defined by $x^R = x_nx_{n-1} \cdots x_2x_1$. We set $\bar{0} = 1$ and $\bar{1} = 0$; moreover, for any string $x = x_1x_2 \cdots x_n$ with $x_i \in \Sigma$, \bar{x} denotes $\bar{x}_1\bar{x}_2 \cdots \bar{x}_n$. To treat a pair of strings, we adopt a *track notation* $\begin{bmatrix} x \\ y \end{bmatrix}$ of [17]. For two symbols σ and τ , the notation $\begin{bmatrix} \sigma \\ \tau \end{bmatrix}$ expresses a new symbol and, for two strings $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_n$ of length n , $\begin{bmatrix} x \\ y \end{bmatrix}$ denotes a string $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \cdots \begin{bmatrix} x_n \\ y_n \end{bmatrix}$ of length n . Since this notation can be seen as a column vector of dimension 2, we can extend it to a *k-track notation*, denoted conveniently by $[x_1, x_2, \dots, x_k]^T$, where “ T ” indicates a transposed vector.

A collection of strings over Σ is a *language* over Σ . A set Σ^k , where $k \in \mathbb{N}$, consists only of strings of length k . In particular, Σ^0 indicates the set $\{\lambda\}$. The *Kleene closure* Σ^* of Σ is the infinite union $\bigcup_{k \in \mathbb{N}} \Sigma^k$. Similarly, the notation $\Sigma^{\leq k}$ is used to mean $\bigcup_{i=1}^k \Sigma^i$. Given a language A over Σ , its *complement* is $\Sigma^* - A$, which

is also denoted by \overline{A} as long as the underlying alphabet Σ is clear from the context. We use the following three class operations between two language families \mathcal{C}_1 and \mathcal{C}_2 : $\mathcal{C}_1 \wedge \mathcal{C}_2 = \{A \cap B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, $\mathcal{C}_1 \vee \mathcal{C}_2 = \{A \cup B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, and $\mathcal{C}_1 - \mathcal{C}_2 = \{A - B \mid A \in \mathcal{C}_1, B \in \mathcal{C}_2\}$, where A and B must be defined over the same alphabet.

As our basic computation models, we use the following types of finite-state machines: *one-way deterministic finite automaton* (or dfa, in short) with λ -moves, *one-way nondeterministic pushdown automaton* (or npda) with λ -moves, and *one-way probabilistic pushdown automaton* (or ppda), where a λ -move (or a λ -transition) is a transition of the machine's configurations in which a target tape head stays still. Notice that allowing λ -moves in any computation of a one-way pushdown automaton is crucial when output tapes are particularly involved.

Formally, an npda M is a tuple $(Q, \Sigma, \{\$, \#\}, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$, where Q is a finite set of inner states, Σ is an input alphabet, Γ is a stack alphabet, $Z_0 (\in \Gamma)$ is the bottom marker of a stack, $q_0 (\in Q)$ is the initial state, $Q_{acc} (\subseteq Q)$ is a set of accepting states, $Q_{rej} (\subseteq Q)$ is a set of rejecting states, and δ is a transition function mapping $(Q - Q_{halt}) \times (\tilde{\Sigma} \cup \{\lambda\}) \times \Gamma$ to $\mathcal{P}(Q \times \Gamma^*)$ with $\tilde{\Sigma} = \Sigma \cup \{\$, \#\}$ and $Q_{halt} = Q_{acc} \cup Q_{rej}$. Any step associated with an application of transition of the form $\delta(q, \lambda, a)$ is called a λ -move (or a λ -transition). The machine M is equipped with a read-only input tape and its tape head cannot move backward. On such a read-only input tape, an input string is surrounded by two distinguished endmarkers ($\$$ and $\#$) and, as soon as a tape head steps outside of these endmarkers, the machine is thought to be *aborted*. The machine must halt instantly after entering a *halting state* (i.e., either an accepting state or a rejecting state). More importantly, we may not be able to implement an internal clock inside an npda to measure its runtime. Therefore, we need to demand that *all* computation paths of M should terminate *eventually*; in other words, along any computation path, M must enter a halting state to stop. For any of the above machines M , we write $PATH_M(x)$ to express a collection of all computation paths produced by M on input x and we use $ACC_M(x)$ to denote a set of all accepting computation paths of M on input x .

Whenever we refer to a *write-only tape*, we always assume that (i) initially, all cells of the tape are blank, (ii) a tape head starts at the so-called *start cell*, (iii) the tape head steps forward whenever it writes down any non-blank symbol, and (iv) the tape head can stay still only in a blank cell. Therefore, all cells through which the tape head passes during a computation must contain no blank symbols. An *output* (outcome or output string) along a computation path is a string produced on the output tape after the computation path is terminated. We call an output string *valid* (or legitimate) if it is produced along a certain accepting computation path. When we refer to the machine's outputs, we normally disregard any strings left on the output tape on a rejecting computation path.

The notations REG, CFL, and DCFL stand for the families of all regular languages, of all context-free languages, and of all deterministic context-free languages, respectively. An advised language family REG/ n in [17] consists of languages L such that there exist an advice alphabet Γ , a length-preserving (total) advice function $h : \mathbb{N} \rightarrow \Gamma^*$, and a language $A \in \text{REG}$ satisfying $L = \{x \mid [h(\frac{x}{|x|})] \in A\}$, where h is *length preserving* if $|h(n)| = n$ for all numbers $n \in \mathbb{N}$. By replacing REG with CFL in REG/ n . Another advised family CFL/ n in [20] is obtained from CFL. A language over a single-letter alphabet is called *tally* and the notation TALLY indicates the collection of all tally languages.

A *multi-valued partial function* f is in CFLMV if there exists an npda M equipped with a one-way read-only input tape together with a write-only output tape such that, for every string x , $f(x)$ is a set composed of all outcomes of N on the input x along accepting computation paths [23].

2.2 Circuit Families and Higher Complexity Classes

For higher complexity classes, we will review basic notions and notations. To handle time/spec-bounded computation, we use two models of *two-way deterministic Turing machine* (or DTM) and *two-way nondeterministic Turing machine* (or NTM). Each of those machines is conventionally equipped with a single read/write input/work tape unless otherwise stated for clarity. Let P (resp., NP) be composed of all languages recognized by DTMs (resp., NTMs) in polynomial time. Given each index $k \in \mathbb{N}$, we define $\Delta_0^P = \Delta_1^P = \Sigma_0^P = \Pi_0^P = P$, $\Delta_{k+1}^P = P^{\Sigma_k^P}$, $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$, and $\Pi_{k+1}^P = \text{co-}\Sigma_{k+1}^P$, where $P^C = \bigcup_{A \in \mathcal{C}} P^A$ (resp., $NP^C = \bigcup_{A \in \mathcal{C}} NP^A$) and P^A (resp., NP^A) is the family of languages recognized by polynomial-time DTMs (resp., NTMs) with adaptive queries to a set A , which is given as an *oracle*. Those language families constitute the so-called *polynomial(-time) hierarchy* [15, 16]. Let $\text{PH} = \bigcup_{k \in \mathbb{N}} \Sigma_k^P$. We denote by L the family of all languages, each of which is recognized by a certain DTM with a two-way read-only input tape and a two-way read/write work tape using $O(\log n)$ cells of the work tape.

Let 1-DLIN consist of languages recognized by one-tape linear-time DTMs. For the precise definition of and discussion on a one-tape linear-time deterministic computation, the reader refers to [17].

For each fixed constant $k \in \mathbb{N}$, NC^k expresses a collection of languages recognized by log-space uniform Boolean circuits of polynomial-size and $O(\log^k n)$ -depth. It is known that NC^0 is properly included within NC^1 ; however, no other separations are known to date. Similarly, AC^k is defined except that all Boolean gates in a circuit may have unbounded fan-in. Moreover, SAC^1 demotes a class of languages recognized by log-space uniform families of polynomial-size Boolean circuits of $O(\log n)$ depth and *semi-bounded* fan-in (that is, having *AND* gates of bounded fan-in and *OR* gates of unbounded fan-in), provided that the negations appear only at the input level. This class SAC^1 is located between NC^1 and NC^2 . Venkateswaran [18] demonstrated that the family of languages log-space many-one reducible to context-free languages characterizes SAC^1 . Moreover, TC^1 consists of all languages recognized by log-space uniform families of $O(\log n)$ -depth polynomial-size circuits whose gates compute *threshold functions*.

3 Nondeterministic Reducibilities

A typical way of comparing the computational complexity of two formal languages is various forms of *resource-bounded reducibility* between them. Such reducibility is also regarded as a *relativization* of its underlying language family. Hereafter, we intend to introduce an appropriate notion of *nondeterministic many-one reducibility* to a theory of context-free languages using a specific computation model of one-way nondeterministic pushdown automata (or npda's). This new reducibility catapults a basic architecture of a hierarchy built over the family CFL of context-free languages in Section 4.

3.1 Many-One Reductions by Npda's

Our exposition begins with an introduction of an appropriate form of nondeterministic many-one reducibility whose reductions are operated by npda's. In the past literature, there were preceding ground works on many-one reducibilities within a framework of a theory of formal languages and automata. Based on deterministic/nondeterministic finite automata (or dfa's/nfa's), for instance, Reinhardt [14] discussed two many-one reducibilities between two languages. Tadaki, Yamakami, and Li [17] also studied the roles of various many-one reducibilities defined by one-tape linear-time Turing machines, which turn out to be closely related to finite automata. Notice that those computation models have no extra memory storage to use. In contrast, we attempt to use npda's as a basis of our reducibility.

An *m-reduction machine* is an npda equipped with an extra *query tape* on which the machine writes a string surrounded by blank cells starting at the designated *start cell* for the purpose of a query to a given *oracle*. We treat the query tape as an output tape, and thus the query-tape head must move to a next blank cell whenever it writes a non-blank symbol. Formally, an *m-reduction machine* is a tuple $(Q, \Sigma, \{\epsilon, \$\}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$, where Θ is a query alphabet and δ is now of the form

$$\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Gamma \rightarrow \mathcal{P}(Q \times (\Theta \cup \{\lambda\}) \times \Gamma^*).$$

There are two types of λ -moves. Assuming $(p, \tau, \xi) \in \delta(q, \sigma, \gamma)$, if $\sigma = \lambda$, then the input-tape head stays still (or makes a λ -move); on the contrary, if $\tau = \lambda$, then the query-tape head stays still (or makes a λ -move).

We say that a language L over alphabet Σ is *many-one CFL-reducible* to another language A over alphabet Γ if there exists an *m-reduction machine* M using Σ and Γ respectively as the input alphabet and the query alphabet such that, for every input $x \in \Sigma^*$, (1) along each computation path $p \in \text{ACC}_M(x)$, M produces a valid query string $y_p \in \Gamma^*$ on the query tape and (2) $x \in L$ if and only if $y_p \in A$ for an appropriate computation path $p \in \text{ACC}_M(x)$. For simplicity, we also say that M reduces (or *m-reduces*) L to A . In other words, L is many-one CFL-reducible to A if and only if there exists a multi-valued partial function $f \in \text{CFLMV}$ satisfying $L = \{x \mid f(x) \cap A \neq \emptyset\}$. With the use of this new reducibility, we make the notation CFL_m^A denotes the family of all languages L that are many-one CFL-reducible to A , where the language A is customarily called an *oracle*. Making an analogy with “oracle Turing machine” that functions as a mechanism of reducing languages to A , we want to use the term “oracle npda” to mean an npda that is equipped with an extra write-only output tape (called a *query tape*) besides a read-only input tape. Given an oracle npda M and an oracle A , the notation $L(M, A)$ (or $L(M^A)$) denotes the set of strings accepted by M relative to A .

Let us start with a quick example of languages that are many-one CFL-reducible to languages in CFL.

Example 3.1 As the first example, setting $\Sigma = \{0, 1\}$, let us consider the language $Dup_2 = \{xx \mid x \in \Sigma^*\}$. This language is known to be non-context-free (see, e.g., [7, 11]); however, it can be many-one CFL-reducible to CFL, because an m -reduction machine nondeterministically produces a query word $x^R \sharp y$ from every input of the form xy using a stack appropriately, and a CFL-oracle checks whether $x = y$ from the input $x^R \sharp y$ using its own stack. In other words, Dup_2 belongs to CFL_m^{CFL} . Similarly, the non-context-free language $Dup_3 = \{xxx \mid x \in \Sigma^*\}$ also falls into CFL_m^{CFL} . For this case, we design a reduction machine to produce $x^R \sharp y \sharp z$ from each input xyz and make an oracle check whether $x = y = z$ by using its stack twice. These examples prove that $CFL_m^{CFL} \neq CFL$. A similar language $Match = \{x \# w \mid \exists u, v [w = uv]\}$, where $\#$ is a separator not in x and w , also belongs to CFL_m^{CFL} .

Unlike polynomial-time many-one reducibility, our many-one CFL-reducibility does not, in general, satisfy the *transitivity property*, in which $A \in CFL_m^B$ and $B \in CFL_m^C$ imply $A \in CFL_m^C$ for any languages A, B, C . To prove this fact, notice that, if CFL is closed under the reducibility, then $CFL_m^{CFL} = CFL$ must hold; however, this contradicts what we have seen in Example 3.1. This non-closure property certainly marks a critical feature of the computational behaviors of languages in CFL. In what follows, we will slightly strengthen this separation between CFL_m^{CFL} and CFL even in the presence of advice.

Proposition 3.2 $CFL_m^{CFL} \not\subseteq CFL/n$.

To show this separation, we will briefly review a notion of k -conjunctive closure over CFL. Given each number $k \in \mathbb{N}^+$, the k -conjunctive closure of CFL, denoted $CFL(k)$ in [23], is defined recursively as follows: $CFL(1) = CFL$ and $CFL(k+1) = CFL(k) \wedge CFL$. These language families truly form an infinite hierarchy [12]. For convenience, we set $CFL(\omega) = \bigcup_{k \in \mathbb{N}^+} CFL(k)$. For known advised language families, it holds that $CFL \not\subseteq REG/n$ [17], $co-CFL \not\subseteq CFL/n$ [20], and $CFL(2) \not\subseteq CFL/n$ [21]. In the following proof of Proposition 3.2, we attempt to prove that $CFL(2) \subseteq CFL_m^{CFL}$ and $CFL(2) \not\subseteq CFL/n$.

Proof of Proposition 3.2. Toward a contradiction, we assume that $CFL_m^{CFL} \subseteq CFL/n$. In this proof, we need the inclusion relation $CFL(2) \subseteq CFL_m^{CFL}$. As for a later reference, we intend to prove a more general statement below.

Claim 1 For every index $k \geq 1$, $CFL(k+1) \subseteq CFL_m^{CFL(k)}$.

Proof. Let L be any language in $CFL(k+1)$ and take two languages $L_1 \in CFL$ and $L_2 \in CFL(k)$ for which $L = L_1 \cap L_2$. There exists an npda M_1 that recognizes L_1 . Without loss of generality, we assume that M_1 enters a final state (either an accepting state or a rejecting state) when it scans the right endmarker $\$$. Now, a new oracle npda N is defined to behave as follows. On input x , N starts simulating M_1 on x . While reading each symbol from x , N also copies it down to a write-only query tape. When M_1 halts in a final state, N enters the same inner state. It holds that, for any input x , x is in L if and only if N on the input x produces the query string x in an accepting state and x is actually in L_2 . This equivalence implies that L belongs to $CFL_m^{L_2}$, which is a subclass of $CFL_m^{CFL(k)}$. \square

Since $CFL(2) \subseteq CFL_m^{CFL}$ by Claim 1, our assumption implies that $CFL(2) \subseteq CFL/n$. This contradicts the class separation $CFL(2) \not\subseteq CFL/n$, proven in [21]. Therefore, the proposition holds. \square

A *Dyck language* L over alphabet $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_d\} \cup \{\sigma'_1, \sigma'_2, \dots, \sigma'_d\}$ is a language generated by a deterministic context-free grammar whose production set is $\{S \rightarrow \lambda | SS | \sigma_i S \sigma'_i : i \in [d]\}$, where S is a start symbol. For convenience, denote by $DYCK$ the family of all Dyck languages.

Lemma 3.3 $CFL_m^{CFL} = CFL_m^{DCFL} = CFL_m^{DYCK}$.

In the following proof, we will employ a simple but useful technique of guessing and verifying a correct *stack history* (namely, a series of consecutive stack transitions). Whenever an oracle npda tries to either push symbols into its stack or pop a symbol from the stack, instead of actually using the stack, we write its stack transition down on a query tape and ask an oracle to verify that it is indeed a correct stack history. This technique will be frequently used in other sections.

Proof of Lemma 3.3. Since $CFL_m^{DYCK} \subseteq CFL_m^{DCFL} \subseteq CFL_m^{CFL}$ trivially holds, we are hereafter focused on proving that $CFL_m^{CFL} \subseteq CFL_m^{DYCK}$. As the first step toward this goal, we will prove the following characterization of CFL in terms of Dyck languages. Notice that Reinhardt [14] proved a similar statement

using a language called L_{pp} .

Claim 2 $\text{CFL} = \text{NFA}_m^{DYCK}$.

Proof. (\subseteq) For any language L in CFL, consider an npda M that recognizes L . Let $\Gamma = \{\sigma_1, \sigma_2, \dots, \sigma_d\}$ be a stack alphabet of M_1 . Corresponding to each symbol σ_i , we introduce another fresh symbol σ'_i and then we set $\Gamma' = \{\sigma'_1, \sigma'_2, \dots, \sigma'_d\}$. Without loss of generality, it is possible to assume that M makes no λ -move until its tape head scans the right end-marker $\$$ (see, e.g., [7] for the proof). For convenience, we further assume that, when the tape head reaches $\$$, M must make a series of λ -moves to empty the stack before entering a halting state. Let us construct a new oracle npda N . In the following description of N , we will intentionally identify all symbols in Γ with “pushed down” symbols, and all symbols in Γ' with “popped up” symbols. Given any input string x , N simulates each step of M ’s computation made on the input x . At a certain step, when M pushes a string $w = \sigma_{i_1}\sigma_{i_2}\dots\sigma_{i_k} \in \Gamma^*$ in place of the top symbol of the stack, N first guesses this top symbol, say, σ_j and then writes down $\sigma'_j w$ on its query tape. This is because σ_j is on the top of the stack and thus it must be first popped up before pushing w . When M pops up a symbol, N guesses it, say, σ_i and writes down σ'_i (not σ_i) on the query tape. Finally, B is chosen to be a Dyck language over the alphabet $\Gamma \cup \Gamma'$. Note that, if a query word w encodes a series of correct stack transitions (with the stack becoming empty when the machine halts), then w obviously belongs to B . Therefore, it holds that L is in CFL_m^B , which is a subclass of CFL_m^{DYCK} .

(\supseteq) Assume that $L \in \text{NFA}_m^B$ for an appropriate oracle B in $DYCK$. Let us take an m -reduction machine M_1 that reduces L to B . Since B is in DCFL (\subseteq CFL), take a deterministic pushdown automaton (or a dpda) M_2 that recognizes B . As before, we assume that M_2 makes no λ -move. We will simulate both M_1 and M_2 by a certain npda N in the following fashion. Given any input x , N starts the simulation of M_1 on x without using a stack. If M_1 tries to write a symbol, say, τ on a query tape, then N instead simulates one step of M_2 ’s computation that corresponds to the scanning of τ together with a certain symbol on the top of the stack. It is not difficult to show that N accepts x if and only if x is in L . Therefore, it follows that $L \in \text{CFL}$. \square

Here, we claim the following equality.

Claim 3 $\text{CFL}_m^A = \text{CFL}_m(\text{NFA}_m^A)$ for any oracle A .

Proof. (\subseteq) This is rather trivial by choosing an appropriate oracle nfa.

(\supseteq) Assume that $L \in \text{CFL}_m^B$ for a certain language B in NFA_m^A . Let us take an oracle npda M_1 recognizing L relative to B and an oracle nfa M_2 recognizing B relative to A . A new machine N is defined to behave as follows. On input x , N simulates M_1 on x . Whenever M_1 tries to write a symbol, say, σ , on a query tape, since M_2 has no stack usage, N can simulate one or more steps (including all possible λ -moves) made by M_2 while it scans σ . Finally, N produces a query word exactly as M_2 does. It is possible to show that N is an oracle npda that recognizes L using A as an oracle. Thus, we obtain the desired membership $L \in \text{CFL}_m^A$. \square

By combining Claims 2 and 3, it follows that $\text{CFL}_m^{\text{CFL}} = \text{CFL}_m(\text{NFA}_m^{DYCK}) \subseteq \text{CFL}_m^{DYCK}$. \square

We will introduce another technique of simulating two or more tape heads moving at (possibly) different speeds by a single tape head. Let us consider an npda M with a write-only output tape. Since the tape heads of M may stay still at any moments (by making λ -moves) on both input and output tapes, it seems difficult to synchronize the moves of those two heads so that we can split the output tape into two tracks and produce a string $\begin{bmatrix} x \\ y \end{bmatrix}$ from input string x and output string y of M . The best we can do is to insert a fresh symbol, say, \natural between input symbols as well as output symbols to adjust the speeds of two tape heads. For this purpose, it is useful to introduce a terminology to describe strings obtained by inserting \natural . Assuming that $\natural \notin \Sigma$, a \natural -extension of a given string x over Σ is a string \tilde{x} over $\Sigma \cup \{\natural\}$ satisfying that x is obtained directly from \tilde{x} simply by removing all occurrences of \natural in \tilde{x} . For instance, if $x = 01101$, then \tilde{x} may be $01\natural1\natural01$ or $011\natural\natural01\natural$.

Naturally, we can extend Dyck languages by adding a special symbol \natural as a part of its underlying alphabet and considering d -tuples of strings over this extended alphabet. More formally, for each index $d \in \mathbb{N}^+$, $DYCK_d^{ext}$ consists of all languages L such that there exist d extended Dyck languages A_1, A_2, \dots, A_d for which L consists of elements of the form $[x_1, x_2, \dots, x_d]^T$ satisfying the following: for every index $i \in [d]$, x_i belongs to A_i . In particular, when $d = 2$, any language L in $DYCK_2^{ext}$ has the form $\{\begin{bmatrix} x \\ y \end{bmatrix} \mid x \in A, y \in B\}$ for certain extended Dyck languages A and B . It is worth noting that $DYCK_d^{ext}$ is a subclass of $\text{DCFL}(d)$.

($= \bigwedge_{i \in [d]} \text{DCFL}$).

The following corollary generalizes Claim 2.

Corollary 3.4 *For each fixed index $d \in \mathbb{N}^+$, $\text{CFL}(d) = \text{NFA}_m^{\text{DYCK}_d^{\text{ext}}}$.*

In the proof of Corollary 3.4, to simplify the description of simulations of given oracle npda's, we need to introduce a special terminology. Let M be any oracle npda and A be any oracle. We say that a string w of the form $[\frac{\tilde{x}}{\tilde{y}}]$ *encodes input x and query word y along a computation path of M* if (i) along a certain accepting computation path p of M on x , M starts with the input x and produces a string y on its query tape, (ii) \tilde{x} and \tilde{y} are \natural -extensions of x and y , respectively, and (iii) there is another oracle npda N that takes w and, by scanning each symbol in w by a single tape head, it can simulate the computation path p as follows. When scanning a symbol of the form $[\frac{\sigma}{\tau}]$, if $\sigma \neq \natural$, then N simulates one step of M while scanning σ on its input tape; on the contrary, if $\sigma = \natural$, then N simulates one λ -move of M without reading any input symbol. At the same time, if $\tau \neq \natural$, then N simulates one step of M while writing τ on its query tape; otherwise, N does nothing. Similarly, we define the concept of " $[\frac{\tilde{y}}{\tilde{z}}]$ encodes stack history y and query word z along a computation path."

Proof of Corollary 3.4. Let L be any language defined as $L = \bigcap_{i \in [d]} A_i$ for k languages A_1, A_2, \dots, A_k in CFL. For each index $i \in [d]$, assume that an npda M_i recognizes A_i . Consider the machine N that, on input x , simulates several steps of M_1, M_2, \dots, M_d in parallel while they scan each input symbol. During this simulation, N writes a stack history of M_i onto the i th track of its query tape. However, to adjust the speeds of d tape heads, we appropriately insert the symbol \natural . If a query word w correctly represent d stack histories of d machines, then w must be in $\text{DYCK}_d^{\text{ext}}$. \square

For later use, we will generalize an argument used in the proof of Claim 2. We say that a language family \mathcal{C} is \natural -*extendible* if, for every language A in \mathcal{C} , two special languages $A_1^\natural = \{[\frac{\tilde{y}}{\tilde{z}}] \mid z \in A\}$ and $A_2^\natural = \{[\frac{\tilde{y}}{\tilde{z}}] \mid y \in A\}$ also belong to \mathcal{C} , where \tilde{y} and \tilde{z} are any \natural -extensions of y and z , respectively.

Lemma 3.5 *Let \mathcal{C} be any nonempty language family. If \mathcal{C} is \natural -extendible, then $\text{CFL}_m^{\mathcal{C}} \subseteq \text{NFA}_m^{\text{DCFL} \wedge \mathcal{C}}$ holds.*

Proof. Take any oracle A in \mathcal{C} and consider any language L in CFL_m^A . Moreover, let M be any m -reduction machine that reduces L to A . With a similar construction as in the proof of Corollary 3.4, we will construct an oracle npda N that behaves as follows. On input x , N simulates M on x and produces on its own query tape strings of the form $[\frac{\tilde{y}}{\tilde{z}}]$ that encode stack history y and query word z along a computation path of M . Choose a Dyck language D that correctly represents any stack histories of M and then define B as the set $\{[\frac{\tilde{y}}{\tilde{z}}] \mid y \in D, z \in A\}$. It is clear by its definition that B belongs to $\text{DCFL} \wedge \mathcal{C}$. \square

Hereafter, we will explore the properties of $\text{CFL}_m^{\text{CFL}(k)}$. First, we will see a few examples.

Example 3.6 The language $Sq = \{0^n 1^{n^2} \mid n \geq 1\}$ belongs to $\text{CFL}_m^{\text{CFL}(3)}$. To see this fact, let us consider the following oracle npda N and oracle A . Given any input w , N first checks if w is of the form $0^i 1^j$. Simultaneously, N nondeterministically selects (j_1, j_2, \dots, j_k) satisfying $j = j_1 + j_2 + \dots + j_k$, and it produces on its query tape a string w' of the form $0^i \natural^{j_1} \natural^{j_2} \dots \natural^{j_k}$. An oracle A receives w' and checks if the following three conditions are all met: (i) $j_1 = j_2, j_3 = j_4, \dots$, (ii) $j_2 = j_3, j_4 = j_5, \dots$, and (iii) $i = k$ by first pushing 0^i into a stack and then counting the number of \natural . It is rather easy to show that A belongs to CFL(3). Therefore, Sq is in CFL_m^A , which is included in $\text{CFL}_m^{\text{CFL}(3)}$. A similar idea proves that the language $Comp = \{0^n \mid n \text{ is a composite number}\}$ belongs to $\text{CFL}_m^{\text{CFL}(2)}$. In symmetry, $Prim = \{0^n \mid n \text{ is a prime number}\}$ is a member of $\text{co}(\text{CFL}_m^{\text{CFL}(2)})$.

The lack of the transitivity property of the many-one CFL-reducibility necessitates an introduction of a helpful abbreviation of a *k-fold application of the reductions*. For any given oracle A , we recursively set $\text{CFL}_{m[1]}^A = \text{CFL}_m^A$ and $\text{CFL}_{m[k+1]}^A = \text{CFL}_m(\text{CFL}_{m[k]}^A)$ for each index $k \in \mathbb{N}^+$. Given each language family \mathcal{C} , the notation $\text{CFL}_{m[k]}^{\mathcal{C}}$ denotes the union $\bigcup_{A \in \mathcal{C}} \text{CFL}_{m[k]}^A$. A close relationship between $\text{CFL}(k)$'s and $\text{CFL}_{m[k]}^{\text{CFL}}$'s is exemplified below.

Theorem 3.7 *For every index $k \in \mathbb{N}^+$, $\text{CFL}_m^{\text{CFL}(k)} = \text{CFL}_{m[k]}^{\text{CFL}}$.*

As an immediate consequence of Theorem 3.7, the union $\bigcup_{k \in \mathbb{N}^+} \text{CFL}_{m[k]}^{\text{CFL}}$ has a succinct expression of

$\text{CFL}_m^{\text{CFL}(\omega)}$.

Corollary 3.8 $\text{CFL}_m^{\text{CFL}(\omega)} = \bigcup_{k \in \mathbb{N}^+} \text{CFL}_{m[k]}^{\text{CFL}}$.

Proof. Since $\text{CFL}_{m[k]}^{\text{CFL}} = \text{CFL}_m^{\text{CFL}(k)}$ by Theorem 3.7, it holds that $\bigcup_{k \in \mathbb{N}^+} \text{CFL}_{m[k]}^{\text{CFL}} = \bigcup_{k \in \mathbb{N}^+} \text{CFL}_m^{\text{CFL}(k)}$. Thus, it suffices to show that $\bigcup_{k \in \mathbb{N}^+} \text{CFL}_m^{\text{CFL}(k)} = \text{CFL}_m^{\text{CFL}(\omega)}$. Since $\text{CFL}(k) \subseteq \text{CFL}(\omega)$, we obtain $\bigcup_{k \in \mathbb{N}^+} \text{CFL}_m^{\text{CFL}(k)} \subseteq \bigcup_{k \in \mathbb{N}^+} \text{CFL}_m^{\text{CFL}(\omega)}$. The last term coincides with $\text{CFL}_m^{\text{CFL}(\omega)}$ since $\text{CFL}_m^{\text{CFL}(\omega)}$ is independent of the value k . Conversely, let L be any language in $\text{CFL}_m^{\text{CFL}(\omega)}$. Take an appropriate oracle $B \in \text{CFL}(\omega)$ for which $L \in \text{CFL}_m^B$. Since $\text{CFL}(\omega) = \bigcup_{k \in \mathbb{N}^+} \text{CFL}(k)$, B must be in $\text{CFL}(k)$ for an appropriate index $k \in \mathbb{N}^+$. This implies that $L \in \text{CFL}_m^B \subseteq \text{CFL}_m^{\text{CFL}(k)}$. Therefore, it holds that $\text{CFL}_m^{\text{CFL}(\omega)} \subseteq \bigcup_{k \in \mathbb{N}^+} \text{CFL}_m^{\text{CFL}(k)}$. This completes the proof of the corollary. \square

Now, we are focused on the proof of Theorem 3.7. When $k = 1$, it holds that $\text{CFL}_{m[1]}^{\text{CFL}} = \text{CFL}_m^{\text{CFL}(1)} = \text{CFL}_m^{\text{CFL}}$. The proof of Proposition 3.7 for $k \geq 2$ is made up of two lemmas, Lemmas 3.9 and 3.10.

Lemma 3.9 For every index $k \geq 2$, $\text{CFL}_{m[k]}^{\text{CFL}} \subseteq \text{CFL}_m^{\text{CFL}(k)}$ holds.

Proof. Let us consider the case where $k = 2$. First, we claim the following inclusion relationship.

Claim 4 For every index $r \in \mathbb{N}^+$, $\text{CFL}_{m[2]}^{\text{CFL}(r)} \subseteq \text{CFL}_m^{\text{CFL}(r) \wedge \text{CFL}}$.

Proof. For a certain language $A \in \text{CFL}(r)$, let us assume that $L \in \text{CFL}_{m[2]}^A$. Furthermore, choose an appropriate set B and let two m -reduction machines M_1 and M_2 respectively witness the membership relations $L \in \text{CFL}_m^B$ and $B \in \text{CFL}_m^A$. We will define a new oracle npda N in part using a stack-history technique shown in the proof of Claim 2. Let Γ be a stack alphabet of M_2 . Corresponding to Γ , we prepare an associated alphabet $\Gamma' = \{\sigma' \mid \sigma \in \Gamma\}$ and set $\Gamma \cup \Gamma'$ to be a new stack alphabet. On input x , N simulates M_1 on x in the following way. Whenever M_1 tries to write a symbol, say, b on a query tape, N instead simulates, without using any actual stack, several steps (including λ -moves) of M_2 that can be made during reading b . When M_2 tries to push down a string s by substituting a top symbol of its stack, N guesses this top symbol, say, σ and then produces σ' s on the *upper track* of its query tape. When M_2 pops a symbol, N guesses this popped symbol, say, σ and writes σ' on the *upper track* of the query tape. At the same time during the simulation, N produces M_2 's query word on the *lower track* of the query tape. To fill the idling time of tape heads, we need to insert an appropriate number of symbols \natural so that $\begin{bmatrix} \tilde{y} \\ \tilde{z} \end{bmatrix}$ encodes stack history y and query word z along a computation path of M_2 . Finally, we define C as a collection of strings of the form $\begin{bmatrix} \tilde{y} \\ \tilde{z} \end{bmatrix}$ such that \tilde{y} (resp., \tilde{z}) is a \natural -extension of a correct stack history y (resp., a valid query string z in A). Since the above definition requires the correctness of y and z , C belongs to $\text{CFL}(r) \wedge \text{CFL}$. Moreover, for every string x , x is in L if and only if there exists an accepting computation path in $\text{ACC}_N(x)$, along which N produces $\begin{bmatrix} \tilde{y} \\ \tilde{z} \end{bmatrix}$ in C . This relation implies that C is in CFL_m^C . \square

For the case of $k \geq 3$, it holds that $\text{CFL}_{m[k]}^{\text{CFL}} = \text{CFL}_m(\text{CFL}_{m[k-1]}^{\text{CFL}}) \subseteq \text{CFL}_m(\text{CFL}_m^{\text{CFL}(k-1)})$, where the last inclusion comes from the induction hypothesis. Note that $\text{CFL}_{m[k]}^{\text{CFL}} \subseteq \text{CFL}_m(\text{CFL}_m^{\text{CFL}(k-1)}) = \text{CFL}_{m[2]}^{\text{CFL}(k-1)}$. Claim 4 then implies that $\text{CFL}_{m[k]}^{\text{CFL}} \subseteq \text{CFL}_m^{\text{CFL}(k-1) \wedge \text{CFL}} = \text{CFL}_m^{\text{CFL}(k)}$. \square

Lemma 3.10 For every index $k \geq 1$, $\text{CFL}_m^{\text{CFL}(k)} \subseteq \text{CFL}_{m[k]}^{\text{CFL}}$.

Proof. By induction on $k \geq 1$, we will prove the lemma. Since the lemma is trivially true for $k = 1$, let us assume that $k \geq 2$. Since $\text{CFL}(k) \subseteq \text{CFL}_m^{\text{CFL}(k-1)}$ by Claim 1, it instantly follows that $\text{CFL}_m^{\text{CFL}(k)} \subseteq \text{CFL}_m(\text{CFL}_m^{\text{CFL}(k-1)})$. Moreover, because $\text{CFL}_m^{\text{CFL}(k-1)} \subseteq \text{CFL}_{m[k-1]}^{\text{CFL}}$ by our induction hypothesis, we conclude that $\text{CFL}_m^{\text{CFL}(k)} \subseteq \text{CFL}_m(\text{CFL}_{m[k-1]}^{\text{CFL}}) = \text{CFL}_{m[k]}^{\text{CFL}}$. \square

Toward the end of this section, we will make a brief discussion on a relationship between two language families CFL/n and $\text{CFL}_m^{\text{TALLY}}$. Given a language A over alphabet Σ , the notation $\text{dense}(A)(n)$ indicates $\|A \cap \Sigma^n\|$ for every length $n \in \mathbb{N}$. Let $\text{DENSE}(f(n))$ be the collection of all languages A such that $\text{dense}(A)(n) \leq f(n)$ holds for all lengths $n \in \mathbb{N}$. Note that $\text{TALLY} \subseteq \text{DENSE}(O(1)) \subseteq \text{SPARSE}$, where $\text{SPARSE} = \text{DENSE}(n^{O(1)})$.

Proposition 3.11 1. $\text{CFL}/n \subseteq \text{CFL}_m^{\text{DENSE}(O(1))}$.
 2. $\text{CFL}_m^{\text{TALLY}} \subseteq \text{CFL}_m^{\text{CFL}(2)}/n$.

Proof. (1) This is rather obvious by taking an advice function h and define $A = \{h(n) \mid n \in \mathbb{N}\}$, which is in $\text{DENSE}(1)$ by the definition.

(2) Let $L \in \text{CFL}_m^A$ for a certain A in TALLY. Let M be a reduction machine that reduces L to A . Without loss of generality, we assume that $A \subseteq \{1\}^*$. For simplicity, we also impose a restriction that $\lambda \notin A$. Next, we will define N_1 as follows. Let $p(n) = an$ be a linear polynomial that bounds the running time of M on inputs of length $n \geq 1$. We define our advice function h as $h(n) = h_1 h_2 \cdots h_n$ for any number $n \in \mathbb{N}^+$, where each symbol h_i equals $[\chi^A(1^{(i-1)a+1}), \chi^A(1^{(i-1)a+2}), \dots, \chi^A(1^{ia})]^T$. Note that $|h(n)| = n$. We then define a language B to satisfy $L = \{x \mid [h(\frac{x}{|x|})] \in B\}$. On input $[\frac{x}{s}]$ with $|s| = |x|$, N_1 simulates M on x and generates a query string $[\frac{\tilde{y}}{\tilde{s}}]$ if M makes a query y , where \tilde{y} and \tilde{s} are \natural -extensions of y and s , respectively. In this process, if $y \notin \{1\}^*$, then N_1 immediately enters a rejecting state. Another oracle npda N_2 works as follows. On input of the form $[\frac{\tilde{y}}{\tilde{s}}]$, using a stack appropriately, N_2 eliminates \natural and produces $y\#s$ on its query tape, where $\#$ is a fresh symbol. The third machine N_3 , taking $y\#s$ as input, finds the i th block $h_i = [b_1, b_2, \dots, b_a]^T$ of s , where $i = \lceil |y|/a \rceil$, and reads a symbol b_j , where $j = |y| - (i-1)a$. If $b_j = 1$, then N_3 enters an accepting state and, otherwise, it enters a rejecting state. This whole process puts B to $\text{CFL}_m(\text{CFL}_m^{\text{CFL}})$, which is $\text{CFL}_{m[2]}^{\text{CFL}}$. By Theorem 3.7, it follows that B is in $\text{CFL}_m^{\text{CFL}(2)}$. It is not difficult to show that, for any string x , $x \in L$ if and only if $[h(\frac{x}{|x|})] \in B$. Therefore, L belongs to $\text{CFL}_m^{\text{CFL}(2)}/n$. \square

3.2 Other Powerful Reducibilities by Npda's

In the previous sections, our primary interest has rested on the many-one CFL-reducibility. It is also possible to introduce two more powerful reducibilities, known as truth-table reducibility and Turing reducibility, into a theory of context-free languages.

We define a notion of *Turing CFL-reducibility* using a model of npda with a write-only query tape and three extra inner states q_{query} , q_{no} , and q_{yes} that represent a query signal and two possible oracle answers, respectively. More specifically, when an oracle npda enters q_{query} , it triggers a query, by which a query word is automatically transferred to an oracle. When an oracle returns its answer, 0 (no) or 1 (yes), the oracle automatically sets the oracle npda's inner state to q_{no} or q_{yes} , respectively. Such a machine is called a *T-reduction machine* (or just an oracle npda as before) and is used to reduce a language to another language. Unlike many-one CFL-reductions, an oracle npda's computation depends on a series of oracle answers. Since an oracle npda, in general, cannot implement any internal clock to control its running time, we should demand that, *no matter what oracle A is provided*, its underlying oracle npda M must halt eventually on *all* computation paths.

Lemma 3.12 For any oracle A , $\text{CFL}_m^A \subseteq \text{CFL}_T^A = \text{CFL}_T^{\bar{A}}$.

Proof. The first inclusion is obvious because the Turing CFL-reducibility can naturally simulate the many-one CFL-reducibility by making a single query at the very end of its computation and deciding to either accept or reject an input based on an oracle answer. To see the last equality, let L be any language in CFL_T^A , witnessed by a T -reduction machine M . Now, we want to prove that $L \in \text{CFL}_T^{\bar{A}}$ via another T -reduction machine N . This machine N is defined to behave as follows. Given any input x , N simulates M on x and, whenever M receives an oracle answer, say, $b \in \{0, 1\}$, N treats it as if \bar{b} and continues the simulation of M . This definition clearly implies that N accepts x relative to \bar{A} if and only if M accepts x relative to A . Thus, L is in $\text{CFL}_T^{\bar{A}}$. We therefore conclude that $\text{CFL}_T^A \subseteq \text{CFL}_T^{\bar{A}}$. By symmetry, we also obtain $\text{CFL}_T^{\bar{A}} \subseteq \text{CFL}_T^A$, implying that $\text{CFL}_T^A = \text{CFL}_T^{\bar{A}}$. \square

As a simple relationship between the Turing and many-one CFL-reducibilities is exemplified in Proposition 3.13. To describe the proposition, we need a notion of the *Boolean hierarchy over CFL*, which was introduced in [24] by setting $\text{CFL}_1 = \text{CFL}$, $\text{CFL}_{2k} = \text{CFL}_{2k-1} \wedge \text{co-CFL}$, and $\text{CFL}_{2k+1} = \text{CFL}_{2k} \vee \text{CFL}$. For simplicity, we denote by BHCFL the union $\bigcup_{k \in \mathbb{N}^+} \text{CFL}_k$. Notice that $\text{CFL} \neq \text{CFL}_2$ holds because $\text{co-CFL} \subseteq \text{CFL}_2$ and $\text{co-CFL} \not\subseteq \text{CFL}$.

Proposition 3.13 $\text{CFL}_T^{\text{CFL}} = \text{CFL}_m^{\text{CFL}_2} = \text{NFA}_m^{\text{CFL}_2}$.

For the proof of this proposition, the following notation is required. If M is an (oracle) npda, then \overline{M} denotes an (oracle) npda obtained from M simply by exchanging between accepting states and rejecting states.

Proof of Proposition 3.13. In this proof, we will demonstrate that (1) $\text{CFL}_T^{\text{CFL}} \subseteq \text{CFL}_m^{\text{CFL}_2}$, (2) $\text{CFL}_m^{\text{CFL}_2} \subseteq \text{NFA}_m^{\text{CFL}_2}$, and (3) $\text{NFA}_m^{\text{CFL}_2} \subseteq \text{CFL}_T^{\text{CFL}}$. If all are proven, then the proposition immediately follows.

(1) We start with an arbitrary language L in CFL_T^A relative to a certain language A in CFL. Take a T -reduction machine M reducing L to A , and let M_A be an npda recognizing A . Hereafter, we will build three new m -reduction machines N_1, N_2, N_3 to show that $L \in \text{CFL}_m^{\text{CFL}_2}$. On input x , the first machine N_1 tries to simulate M on x by running the following procedure. Along each computation path, before M begins producing the i th query word on a query tape, N_1 guesses its oracle answer b_i (either 0 or 1) and writes it down onto its query tape. While M writes the i th query word y_i , N_1 does the same but appends $y_i \#$ to b_i . When M halts, N_1 produces a query word w of the form $b_1 y_1 \# b_2 y_2 \# \cdots \# b_k y_k \#$, where $k \in \mathbb{N}$.

The second machine N_2 works as follows. On input w of the above form, N_2 does the following procedure. On reading b_i , if $b_i = 1$, then N_2 simulates M_A on y_i . If $b_i = 0$, then N_2 skips y_i . Whenever M_A enters a rejecting state, N_2 also enters a rejecting state and halts. The third machine N_3 takes input w and, if $b_i = 0$, then N_3 simulates $\overline{M_A}$ on y_i ; otherwise, N_3 skips y_i . It is not difficult to verify that N_1 m -reduces L to $L(N_2) \wedge \overline{L(N_3)}$. This leads to a conclusion that L is included in $\text{CFL}_m(\text{CFL} \wedge \text{co-CFL}) = \text{CFL}_m^{\text{CFL}_2}$.

(2) Note that CFL_2 is $\#$ -extendible. Proposition 3.5 implies that $\text{CFL}_m^{\text{CFL}_2} \subseteq \text{NFA}_m^{\text{DCFL} \wedge \text{CFL}_2}$. Since $\text{DCFL} \subseteq \text{co-CFL}$, it follows that $\text{DCFL} \wedge \text{CFL}_2 \subseteq \text{co-CFL} \wedge (\text{CFL} \wedge \text{co-CFL}) = (\text{co-CFL} \wedge \text{co-CFL}) \wedge \text{CFL}$. The last term clearly equals $\text{co-CFL} \wedge \text{CFL} = \text{CFL}_2$, and thus we conclude that $\text{CFL}_m^{\text{CFL}_2} \subseteq \text{NFA}_m^{\text{CFL}_2}$.

(3) Choose an oracle A in CFL_2 and consider any language L in CFL_M^A . Furthermore, take two languages $A_1, A_2 \in \text{CFL}$ for which $A = A_1 \cap \overline{A_2}$. Let M be an oracle nfa that recognizes L relative to A . Notice that M has no stack. We will define an oracle npda N as follows. On input x , N first marks 0 on its query tape and start simulating M on x . Whenever M tries to write a symbol σ on its query tape, N writes it down on a query tape and simultaneously copies it into a stack. After M halts with a query word, say, w , N makes the first query with the query word $0w$. If its oracle answer is 0, then N rejects the input. Subsequently, N writes 1 on the query tape (provided that the tape automatically becomes blank), pops the stored string w^R from the stack, and copies it to the query tape. After making the second query with $1w^R$, if its oracle answer equals 1, then N rejects the input. When N has not entered any rejecting state, then N finally accepts the input. The corresponding oracle B is defined as $\{0w \mid w \in A_1\} \cup \{1w^R \mid w \in A_2\}$. It is easy to see that $x \in L$ if and only if N accepts x relative to B . Since CFL is closed under reversal (see, e.g., [7]), $\{1w^R \mid w \in A_2\}$ is context-free, and thus B is in CFL. we then conclude that $L \in \text{CFL}_T^B \subseteq \text{CFL}_T^{\text{CFL}}$. \square

As another typical reducibility, we are focused on *nondeterministic truth-table reducibility*. Notice that an introduction of nondeterministic truth-table reducibility to context-free languages does not seem to be as obvious as that of nondeterministic Turing reducibility. Ladner, Lynch, and Selman [10] first offered such a notion for NP. Another definition, which is apparently weaker than that of Ladner et al., was proposed by Book, Long, and Selman [3] as well as Book and Ko [2]. The following definition follows a spirit of Ladner et al. [10] with a slight twist for its evaluator.

Letting $k \in \mathbb{N}^+$, a language L is in CFL_{ktt}^A if there are a language $B \in \text{REG}$ and an npda N with k write-only output tapes besides a read-only input tape such that, for any input string x , (1) $\text{ACC}_M(x) \neq \emptyset$, (2) along every computation path $p \in \text{ACC}_N(x)$, N produces a string $y_p^{(i)} \in \Gamma^*$ on the i th write-only query tape for each index $i \in [k]$, (3) from a vector $(y_p^{(1)}, y_p^{(2)}, \dots, y_p^{(k)})$ of query words, we generate a k -bit string $z_p = \chi_k^A(y_p^{(1)}, y_p^{(2)}, \dots, y_p^{(k)}) \in \{0, 1\}^k$, and (4) x is in L if and only if $\lfloor \frac{x}{z_p} \rfloor$ is in B for an appropriate computation path $p \in \text{ACC}_N(x)$. Here, the set B is called a *truth table* for A . For convenience sake, we often treat B as a function defined as $B(x, y) = 1$ if $\lfloor \frac{x}{y} \rfloor \in B$ and $B(x, y) = 0$ otherwise. In the end, we set CFL_{btt}^A to be the union $\bigcup_{k \in \mathbb{N}^+} \text{CFL}_{ktt}^A$.

It is clear that $\text{CFL}_m^A \cup \text{CFL}_m^{\overline{A}} \subseteq \text{CFL}_{1tt}^A$. By flipping the outcome (accepting or rejecting) of a computation generated by each dfa that computes a truth-table, we obtain $\text{CFL}_{ktt}^A \subseteq \text{CFL}_{ktt}^{\overline{A}}$. In symmetry, $\text{CFL}_{ktt}^{\overline{A}} \subseteq \text{CFL}_{ktt}^A$ also holds. Therefore, the statement given below holds.

Lemma 3.14 For every language A and index $k \geq 1$, $\text{CFL}_m^A \cup \text{CFL}_m^{\overline{A}} \subseteq \text{CFL}_{ktt}^A = \text{CFL}_{ktt}^{\overline{A}}$.

Unlike NP, we do not know whether $\text{CFL}_{btt}^{\text{CFL}} \subseteq \text{CFL}_T^{\text{CFL}}$ holds. This is mainly because of a restriction

of npda's memory use. It may be counterintuitive that Turing reducibility cannot be powerful enough to simulate truth-table reducibility. On the contrary, the following inclusion holds in a case of a constant number of queries.

Lemma 3.15 *Let $k \in \mathbb{N}^+$ be any constant. For every language A , $\text{CFL}_{kT}^A \subseteq \text{CFL}_{ktt}^A$ holds.*

Proof. Let M be any T -reduction machine that witnesses the membership $L \in \text{CFL}_{kT}^A$. To show that $L \in \text{CFL}_{ktt}^A$, we will build another btt -reduction machine N that is equipped with k query tapes. This oracle npda N works as follows. On input x , it simulates M on the input x . When M tries to write the i th query word, say, y_i , N produces y_i on its i th query tape. The machine N then guesses an oracle answer $a_i \in \{0, 1\}$ for y_i (without any actual query) and remembers the value a_i within the machine's finite control unit. After M halts in an accepting state, N makes k queries to A . Note that M^A accepts x if and only if $a_1 a_2 \cdots a_k = \chi_k^A(y_1, y_2, \dots, y_k)$ along a certain accepting computation path. Thus, N btt -reduces L to A with only k queries. \square

Theorem 3.16 $\text{CFL}_m^{\text{BHCFL}} = \text{CFL}_{btt}^{\text{CFL}} = \text{NFA}_{btt}^{\text{CFL}}$.

Before proving this theorem, we will present a characterization of BHCFL in terms of bounded-truth-table reductions. For this purpose, we need to introduce a new relativization of DFA. A language L is in DFA_{ktt}^A if there exists an oracle dfa that produces k query words y_1, y_2, \dots, y_k such that, for every string x , $x \in L^A$ if and only if $B(x, \chi_k^A(y_1, y_2, \dots, y_k)) = 1$. We set $\text{DFA}_{btt}^A = \bigcup_{k \in \mathbb{N}^+} \text{DFA}_{ktt}^A$ and $\text{DFA}_{btt}^C = \bigcup_{A \in C} \text{DFA}_{btt}^A$ for any language family C .

Lemma 3.17 $\text{BHCFL} = \text{DFA}_{btt}^{\text{CFL}}$.

Proof. We split the lemma into the following two separate claims: (1) $\text{BHCFL} \subseteq \text{DFA}_{btt}^{\text{CFL}}$ and (2) $\text{DFA}_{btt}^{\text{CFL}} \subseteq \text{BHCFL}$.

(1) Our goal is to prove by induction on $k \in \mathbb{N}^+$ that $\text{CFL}_k \subseteq \text{DFA}_{ktt}^{\text{CFL}}$. From this assertion, it follows that $\text{BHCFL} = \bigcup_{k \in \mathbb{N}^+} \text{CFL}_k \subseteq \bigcup_{k \in \mathbb{N}^+} \text{DFA}_{ktt}^{\text{CFL}} = \text{DFA}_{btt}^{\text{CFL}}$.

As for the base case $k = 1$, it is clear that $\text{CFL} \subseteq \text{DFA}_m^{\text{CFL}} \subseteq \text{DFA}_{1tt}^{\text{CFL}}$. Now, let us concentrate on the induction step $k \geq 2$. Meanwhile, we assume that k is even. Since $\text{CFL}_k = \text{CFL}_{k-1} \wedge \text{co-CFL}$, take two languages $L_1 \in \text{CFL}_{k-1}$ and $L_2 \in \text{co-CFL}$ and assume that $L = L_1 \cap L_2$. Assume also that an npda M_2 computes $\overline{L_2}$. Since $L_1 \in \text{DFA}_{(k-1)tt}^{\text{CFL}}$ by our induction hypothesis, there is an oracle dfa, say, M_1 that recognizes L_1 relative to oracle A in CFL. On input x , M_1 produces $(k-1)$ -tuple $(y_1, y_2, \dots, y_{k-1})$ on its query tape and it satisfies that $B(x, \chi_{k-1}^A(y_1, y_2, \dots, y_{k-1})) = 1$ if and only if x is in L_1 . We want to define a new machine N as follows. By simulating M_1 , N generates $k-1$ query words $(y_1, y_2, \dots, y_{k-1})$ as well as a new query word $\natural x$, where \natural is a fresh symbol. Moreover, we define $A' = A \cup \{\natural x \mid M_2 \text{ accepts } x\}$, which is obviously in CFL. Now, we define B' as $\{(x, b_1 b_2 \cdots b_{k-1} b_k) \mid (x, b_1 b_2 \cdots b_{k-1}) \in B \wedge b_k = 0\}$. Clearly, B' is regular since so is B . It also holds that $B'(x, \chi_k^{A'}(y_1, y_2, \dots, y_{k-1}, \natural x)) = 1$ if and only if $B(x, \chi_{k-1}^A(y_1, y_2, \dots, y_{k-1})) = 1$ and $x \in \overline{L_2}$. Therefore, L belongs to $\text{CFL}_{ktt}^{B'} \subseteq \text{CFL}_{ktt}^{\text{CFL}}$.

The case of odd k is proven by slightly modifying the above proof.

(2) We will prove that, for any index $k \in \mathbb{N}^+$, $\text{DFA}_{ktt}^{\text{CFL}} \subseteq \text{CFL}_{k2^{k+1}}$. We begin with the case where $k = 1$. Assume that $L \in \text{DFA}_{1tt}^A$ for a certain language A in CFL. Let M_1 be an oracle npda that recognizes L relative to A and let B be a truth-table used for M_1 . Moreover, let M_2 be an npda that recognizes A . Without loss of generality, we assume that M_2 makes no λ -move. A new machine N_1 works in the following way. Given any input x , N_1 first checks if $B(x, 1) = 1$. Simultaneously, N_1 simulates M_1 on x . When M_1 tries to write a symbol, say, b , on the i th query tape, N_1 simulates one step of M_2 's computation during the scanning of b . Similarly, we define N_0 except that (i) it checks if $B(x, 0) = 1$ and (ii) if M_2 enters an accepting state (resp., a rejecting state), then N_0 enters a rejecting state (resp., an accepting state). It is important to note that this machine N_0 is *co-nondeterministic*, and thus $L(N_0)$ is in co-CFL, whereas $L(N_1)$ belongs to CFL. It also follows that $L = L(N_0) \cup L(N_1)$. Hence, L belongs to $\text{CFL} \vee \text{co-CFL}$, which is included in $\text{CFL}_3 \subseteq \text{CFL}_4$, as requested.

For the case $k \geq 2$, we need to generalize the above argument. Assume that $L \in \text{DFA}_{ktt}^A$ for a certain language A in CFL. Let M_1 be a ktt -reduction machine that reduces L to A and let M_2 be an npda recognizing A . Here, we assume that M_2 makes no λ -move. In the following argument, we fix a string $b = b_1 b_2 \cdots b_k \in \{0, 1\}^k$. Letting $\text{CFL}_2^{(0)} = \text{co-CFL}_2$ and $\text{CFL}_2^{(1)} = \text{CFL}_2$, we define $\text{CFL}_2^{(b_1 b_2 \cdots b_k)}$ as an abbreviation of $\text{CFL}_2^{(b_1)} \vee \text{CFL}_2^{(b_2)} \vee \cdots \vee \text{CFL}_2^{(b_k)}$.

Now, we will introduce two types of machines $N_{b,0}$ and $N_{b,1}$. The machine $N_{b,1}$ takes input x and checks if $B(x, b) = 1$. At the same time, $N_{b,1}$ guesses a number $i \in [k]$ and simulates M_1 on x if $b_i = 1$ (and, otherwise, it enters an accepting state instantly). Whenever M_1 tries to write a symbol, say, σ on its own query tape, $N_{b,1}$ simulates one step of M_2 's computation corresponding to the scanning of σ . As for the other machine $N_{b,0}$, it simulates M_1 on x if $b_i = 0$ (and accepts instantly otherwise). Note that $N_{b,0}$ is a co-nondeterministic machine. For each index $e \in \{0, 1\}$, let $A_{b,e}$ be composed of strings accepted by $N_{b,e}$ and we set $A_b = A_{b,0} \cup A_{b,1}$, which obviously belongs to CFL_2 . It is not difficult to show that $L = \bigcup_{b \in \{0,1\}^k} A_b$; thus, L is in $\bigvee_{b \in \{0,1\}^k} \text{CFL}_2^{(b)}$, which equals $\bigvee_{b \in \{0,1\}^k} [(\bigvee_{i:b_i=1} \text{CFL}_2) \vee (\bigvee_{i:b_i=0} \text{co-CFL}_2)]$. By a simple calculation, the last term coincides with $(\bigvee_{k=1}^{k2^{k-1}} \text{CFL}_2) \vee (\bigvee_{k=1}^{k2^{k-1}} \text{co-CFL}_2)$. Since $\text{co-CFL}_2 = \text{CFL} \vee \text{co-CFL} \subseteq \text{CFL} \vee \text{CFL}_2$, it follows that $\bigvee_{i=1}^{k2^{k-1}} \text{co-CFL}_2 \subseteq \bigvee_{i=1}^{k2^{k-1}} (\text{CFL} \vee \text{CFL}_2) = (\bigvee_{i=1}^{k2^{k-1}} \text{CFL}) \vee (\bigvee_{i=1}^{k2^{k-1}} \text{CFL}_2) = \bigvee_{i=1}^{k2^{k-1}} \text{CFL}_2$. Therefore, L belongs to $\bigvee_{i=1}^{k2^k} \text{CFL}_2$. Note that $\text{CFL}_{k2^{k+1}} = \bigvee_{k=1}^{k2^k} \text{CFL}_2$ by Claim 8. We thus conclude that L is in $\text{CFL}_{k2^{k+1}}$. \square

Finally, we will present the proof of Theorem 3.16. for this proof, we need to introduce the third simulation technique of encoding a computation path of an npda into a string. Notice that a series of nondeterministic choices made by an npda M uniquely specifies which computation path the machine has followed. We encode such a series into a single string. Let δ be a transition function of M . First, we rewrite δ in the following way. If δ has an entry of the form $\delta(q, \sigma, \tau, \eta) = \{(p_1, \xi_1, \zeta_1), (p_2, \xi_2, \zeta_2)\}$, then we split it into two transitions: $\delta(q, \sigma, \tau, \eta) = (p_1, \xi_1, \zeta_1)$ and $\delta(q, \sigma, \tau, \eta) = (p_2, \xi_2, \zeta_2)$. Let D be the collection of all such new transitions. Next, we index all such new transitions using numbers in the integer interval $[\|D\|] = \{1, 2, \dots, \|D\|\}$. A series of transitions can be expressed as a series of those indices, which is regarded as a string over the alphabet $\Sigma = [\|D\|]$. We call such a string an *encoding of a computation path* of M .

Proof of Theorem 3.16. In this proof, we will prove three inclusions: $\text{CFL}_{btt}^{\text{CFL}} \subseteq \text{NFA}_{btt}^{\text{CFL}} \subseteq \text{CFL}_m(\text{DFA}_{btt}^{\text{CFL}}) \subseteq \text{CFL}_{btt}^{\text{CFL}}$. Obviously, these inclusions together ensure the desired equations given in the theorem.

Claim 5 $\text{CFL}_{btt}^{\text{CFL}} \subseteq \text{NFA}_{btt}^{\text{CFL}}$.

Proof. Fix $k \in \mathbb{N}^+$ and let L be any language in CFL_{ktt}^A for a certain oracle $A \in \text{CFL}$. For this L , there is an oracle npda, say, M_1 that recognizes L using A as an oracle. We want to define another machine N by modifying M_1 in a similar way presented in the proof of Claim 2. On input x , N simulates M_1 on x using $k+1$ query tapes as follows. When M_1 tries to push $\sigma_1\sigma_2 \cdots \sigma_n$ in place of a certain symbol on the top of a stack, N guesses this symbol, say, σ and then writes down $\sigma'\sigma_1 \cdots \sigma_n$ on the $k+1$ st query tape. If M_1 pops a certain symbol, then N first guesses this symbol, say, σ and writes down σ' on the $k+1$ st query tape. Finally, we define B' as the set $\{[x, y_1, y_2, \dots, y_k, 1]^T \mid B(x, y_1y_2 \cdots y_k) = 1\}$. Associated with N 's $k+1$ st query, we choose an appropriate language C in $DYCK$. Define $A' = A \cup C$, assuming that C is based on a different alphabet. Note that x is in L if and only if $B'(x, \chi_{k+1}^A(x)) = 1$. Hence, L belongs to $\text{NFA}_{(k+1)tt}^{A'}$, which is a subclass of $\text{NFA}_{btt}^{\text{CFL}}$. \square

Claim 6 $\text{NFA}_{btt}^{\text{CFL}} \subseteq \text{CFL}_m(\text{DFA}_{btt}^{\text{CFL}})$.

Proof. Let $k \geq 1$. Assume that $L \in \text{NFA}_{btt}^A$ with a certain oracle $A \in \text{CFL}$. Let M be a ktt -reduction machine that reduces L to A . To show that $L \in \text{CFL}_m(\text{DFA}_{btt}^{\text{CFL}})$, we define an oracle npda N_1 as follows. Given any input x , N_1 simulates M by writing a string that encodes a computation path y of M . At any time when M tries to write any symbol on its own query tapes, N_1 simply ignores the symbol and continues the above simulation. Next, we define N_2 as follows. On input of the form $[\frac{x}{y}]$, N_2 deterministically simulates M on x by following a series of nondeterministic choices specified by y , and N_2 produces k query strings as M does. Note that N_1 m -reduces L to $L(N_2, A)$ and that $L(N_2, A)$ is in DFA_{btt}^A . Therefore, L belongs to $\text{CFL}_m^{L(N_2, A)} \subseteq \text{CFL}_m(\text{DFA}_{btt}^{\text{CFL}})$. \square

Claim 7 $\text{CFL}_m(\text{DFA}_{btt}^{\text{CFL}}) \subseteq \text{CFL}_{btt}^{\text{CFL}}$.

Proof. Take any oracle $A \in \text{DFA}_{btt}^{\text{CFL}}$ and assume that $L \in \text{CFL}_m^A$. Let M_1 be an m -reduction machine reducing L to A . Fixing $k \in \mathbb{N}^+$, we assume that M_2 is a ktt -reduction machine that reduces A to a certain language B in CFL . Now, we define N as follows. On input x , N simulates M_1 on x . When M_1

writes a symbol, say, b , N simulates one or more steps (including λ -moves) of M_2 's computation during the scanning of b . Finally, N outputs M_2 's k query strings. It holds that N k tt-reduces L to B , and thus L is in $\text{CFL}_{btt}^B \subseteq \text{CFL}_{btt}^{\text{CFL}}$. \square

We have just proven that $\text{CFL}_{btt}^{\text{CFL}} = \text{NFA}_{btt}^{\text{CFL}} = \text{CFL}_m(\text{DFA}_{btt}^{\text{CFL}})$. By Lemma 3.17, the last term equals $\text{CFL}_m^{\text{BHCFL}}$. This completes the proof. \square

3.3 Languages That are Low for CFL

We will briefly discuss a notion of *lowness*, which concerns oracles that contain little information to help underlying oracle machines improve their recognition power. More precisely, consider a language family \mathcal{C} that is *many-one relativizable*. A language A is called *many-one low* for \mathcal{C} if $\mathcal{C}_m^A \subseteq \mathcal{C}$ holds. We define $\text{low}_m\mathcal{C}$ to be the set of all languages that are low for \mathcal{C} ; that is, $\text{low}_m\mathcal{C} = \{A \mid \mathcal{C}_m^A \subseteq \mathcal{C}\}$. Similarly, we define $\text{low}_{btt}\mathcal{C}$ and $\text{low}_T\mathcal{C}$ as a collection of all languages that are “btt low for \mathcal{C} ” and “Turing low for \mathcal{C} ,” respectively.

Languages in REG, when playing as oracles, have no power to increase the computational complexity of relativized CFL.

Lemma 3.18 $\text{CFL} = \text{CFL}_m^{\text{REG}} = \text{CFL}_{btt}^{\text{REG}} = \text{CFL}_T^{\text{REG}}$.

Proof. Since $\text{CFL} \subseteq \text{CFL}_m^{\Sigma^*}$ and $\Sigma^* \in \text{REG}$ for $\Sigma = \{0, 1\}$, it follows that $\text{CFL} \subseteq \text{CFL}_m^{\text{REG}}$. Moreover, by Lemmas 3.12 and 3.14, it holds that $\text{CFL}_m^{\text{REG}} \subseteq \text{CFL}_{btt}^{\text{REG}}$ and $\text{CFL}_m^{\text{REG}} \subseteq \text{CFL}_T^{\text{REG}}$. To show that $\text{CFL}_{btt}^{\text{REG}} \subseteq \text{CFL}$, take any language L in CFL_{ktt}^A for a certain index $k \in \mathbb{N}^+$ and a certain language $A \in \text{REG}$. Let M be an oracle npda equipped with two write-only query tapes that k tt-reduces L to A . In addition, let N denote a dfa recognizing A . We aim at proving that $L \in \text{CFL}$. Let us consider the following algorithm. We start simulating M on each input without using any write-only tapes. When M tries to write down a k -tuple of symbols (s_1, s_2, \dots, s_k) , we instead simulate N using only inner states. Note that we do not need to keep on the query tapes any information on (s_1, s_2, \dots, s_k) . Along each computation path, we accept the input if both M and N enter accepting states. Since the above algorithm requires no query tapes, it can be implemented by a certain npda. Moreover, since the algorithm correctly recognizes L , we conclude that L is in CFL.

Based on a similar idea, we can prove that $\text{CFL}_T^{\text{REG}} \subseteq \text{CFL}$. \square

Lemma 3.19 1. $\text{REG} \subseteq \text{low}_T\text{CFL} \cap \text{low}_{btt}\text{CFL} \subseteq \text{low}_T\text{CFL} \cup \text{low}_{btt}\text{CFL} \subseteq \text{low}_m\text{CFL} \subsetneq \text{CFL}$.

2. $\text{low}_{btt}\text{CFL} \subsetneq \text{CFL} \cap \text{co-CFL}$.

Proof. (1) From Lemma 3.18, it holds that $\text{CFL}_T^{\text{REG}} = \text{CFL}_{btt}^{\text{REG}} = \text{CFL}$. Thus, it follows that $\text{REG} \subseteq \text{low}_T\text{CFL} \cap \text{low}_{btt}\text{CFL}$. The third inclusion comes from the fact that $\text{CFL}_m^A \subseteq \text{CFL}_{btt}^A \cap \text{CFL}_T^A$ for any oracle A . The last inclusion is shown as follows. Take any language A in low_mCFL . This means that $\text{CFL}_m^A \subseteq \text{CFL}$. Since A belongs to CFL_m^A , it holds that $A \in \text{CFL}_m^A \subseteq \text{CFL}$. Next, we wish to show that $\text{CFL} \neq \text{low}_m\text{CFL}$. If $\text{CFL} = \text{low}_m\text{CFL}$ holds, we obtain $\text{CFL}_m^{\text{CFL}} \subseteq \text{CFL}$. Since $\text{CFL}(2) \subseteq \text{CFL}_m^{\text{CFL}}$ by Lemma 1, we immediately conclude $\text{CFL}(2) = \text{CFL}$. This is indeed a contradiction against the well-known result that $\text{CFL}(2) \neq \text{CFL}$. Thus, it must hold that $\text{low}_m\text{CFL} \neq \text{CFL}$.

(2) For this inclusion, let us consider any language A in $\text{low}_{btt}\text{CFL}$; that is, $\text{CFL}_{btt}^A \subseteq \text{CFL}$. Since $A, \bar{A} \in \text{CFL}_{btt}^A$, it follows that $A, \bar{A} \in \text{CFL}$. Thus, A must belong to $\text{CFL} \cap \text{co-CFL}$.

For the last proper inclusion, we assume that $\text{low}_{btt}\text{CFL} = \text{CFL} \cap \text{co-CFL}$. Thus, $\text{CFL}_{btt}^{\text{CFL} \cap \text{co-CFL}} = \text{CFL}$. This implies $\text{CFL}_m^{\text{DCFL}} \subseteq \text{CFL}$, because $\text{CFL}_m^B \subseteq \text{CFL}_{btt}^B$ and $\text{DCFL} \subseteq \text{CFL} \cap \text{co-CFL}$. However, this is a contradiction because $\text{CFL}_m^{\text{DCFL}} \not\subseteq \text{CFL}/n$ by Proposition 3.2 and Lemma 3.3. \square

It is not known whether all inclusion relations in the lemma are actually proper.

4 The CFL Hierarchy

Nondeterministic polynomial-time Turing reductions have been used to build the polynomial hierarchy, each level of which is induced from its lower level by applying the reductions. With use of our Turing CFL-reducibility defined in Section 3.2, a similar construction applied to CFL introduces a unique hierarchy,

which we call the CFL hierarchy. We will explore fundamental properties of this new hierarchy throughout this section.

4.1 Turing CFL-Reducibility and a Hierarchy

Applying Turing CFL-reductions to CFL level by level, we can build a useful hierarchy, called the *CFL hierarchy*, whose k th level consists of three language families Δ_k^{CFL} , Σ_k^{CFL} , and Π_k^{CFL} . To be more precise, for each level $k \geq 1$, we set $\Delta_0^{\text{CFL}} = \Delta_1^{\text{CFL}} = \Sigma_0^{\text{CFL}} = \Pi_0^{\text{CFL}} = \text{DCFL}$, $\Delta_{k+1}^{\text{CFL}} = \text{DCFL}_T(\Sigma_k^{\text{CFL}})$, $\Pi_k^{\text{CFL}} = \text{co-}\Sigma_k^{\text{CFL}}$, and $\Sigma_{k+1}^{\text{CFL}} = \text{CFL}_T(\Sigma_k^{\text{CFL}})$. Additionally, we set $\text{CFLH} = \bigcup_{k \in \mathbb{N}^+} \Sigma_k^{\text{CFL}}$.

The CFL hierarchy can be used to categorize the complexity of typical non-context-free languages discussed in most introductory textbooks, e.g., [7, 11]. We will review typical examples that fall into the CFL hierarchy.

Example 4.1 We have seen in Example 3.1 the languages $\text{Dup}_2 = \{xx \mid x \in \{0,1\}^*\}$ and $\text{Dup}_3 = \{xxx \mid x \in \{0,1\}^*\}$, which are both in $\text{CFL}_m^{\text{CFL}}$. Note that, since $\text{CFL}_m^A \subseteq \text{CFL}_T^A$ for any oracle A , every language in $\text{CFL}_m^{\text{CFL}}$ belongs to $\text{CFL}_T^{\text{CFL}} = \Sigma_2^{\text{CFL}}$. Therefore, Dup_2 and Dup_3 are in Σ_2^{CFL} . In addition, as shown in Example 3.6, the language $Sq = \{0^n 1^{n^2} \mid n \geq 1\}$ is in $\text{CFL}_m^{\text{CFL}(2)}$ while $\text{Prim} = \{0^n \mid n \text{ is a prime number}\}$ is in $\text{co-}(\text{CFL}_m^{\text{CFL}(2)})$. Owing to a proposition (Proposition 4.8) proven later, it holds that $\text{CFL}_m^{\text{CFL}(2)} \subseteq \Sigma_3^{\text{CFL}}$. Therefore, we conclude that Sq is in Σ_3^{CFL} and Prim is in Π_3^{CFL} . A similar but more involved example is the language $\text{MulPrim} = \{0^{mn} \mid m \text{ and } n \text{ are prime numbers}\}$. Consider the following three npda's. The first machine M_1 nondeterministically partitions a given input 0^k into (y_1, y_2, \dots, y_n) and produces $w = y_1 \# y_2 \# \dots \# y_n$ on a query tape by inserting a new symbol $\#$. During this process, M_1 pushes $u = y_1 \# 1^n$ to a stack, where $\#$ is a fresh symbol. In the end, M_1 appends $\#u$ to w on the query tape. In receiving $w \# u$ as an input, the second machine M_2 checks if $y_{2i-1} = y_{2i}$ for $i = 1, 2, \dots$. At any moment when the checking process fails, M_2 enters a rejecting state and halts. Next, M_2 nondeterministically partitions y_1 into (z_1, z_2, \dots, z_e) and 1^n into (x_1, x_2, \dots, x_d) and then it produces $w \# u' \# v'$ on the query tape, where $u' = z_1 \# z_2 \# \dots \# z_e$ and $v' = x_1 \# x_2 \# \dots \# x_d$. In receiving $w \# u' \# v'$, the third machine M_3 checks if $y_{2i+1} = y_{2i+1}$ for $i = 1, 2, \dots$. Whenever this process fails, M_3 instantly halts in a rejecting state. Next, M_3 nondeterministically chooses a bit b . If $b = 0$, then M_3 checks if $z_{2i-1} = z_{2i}$ and also $x_{2i-1} = x_{2i}$ for $i = 1, 2, \dots$; on the contrary, if $b = 1$, then M_3 checks if $z_{2i} = z_{2i+1}$ and $x_{2i} = x_{2i+1}$ for $i = 1, 2, \dots$. If this checking process is successful, then M_3 enters a rejecting state; otherwise, it enters an accepting state. By combining those three machines, MulPrim is shown to belong to $\text{CFL}_m(\text{co-}(\text{CFL}_m^{\text{co-CFL}}))$, which equals Σ_3^{CFL} .

Several basic relationships among the components of the CFL hierarchy are exhibited in the next lemma. More structural properties will be discussed later in Section 4.2.

Lemma 4.2 Let k be any integer satisfying $k \geq 1$.

1. $\text{CFL}_T(\Sigma_k^{\text{CFL}}) = \text{CFL}_T(\Pi_k^{\text{CFL}})$ and $\text{DCFL}_T(\Sigma_k^{\text{CFL}}) = \text{DCFL}_T(\Pi_k^{\text{CFL}})$.
2. $\Sigma_k^{\text{CFL}} \cup \Pi_k^{\text{CFL}} \subseteq \Delta_{k+1}^{\text{CFL}} \subseteq \Sigma_{k+1}^{\text{CFL}} \cap \Pi_{k+1}^{\text{CFL}}$.
3. $\text{CFLH} \subseteq \text{DSPACE}(O(n))$.

Proof. (1) This is a direct consequence of Lemma 3.12 since $\Pi_k^{\text{CFL}} = \text{co-}\Sigma_k^{\text{CFL}}$. The case of Turing DCFL-reduction is similar in essence.

(2) Let $k \geq 1$. Since $A \in \text{DCFL}_T^A$ holds for any oracle A , it holds that $\Sigma_k^{\text{CFL}} \subseteq \text{DCFL}_T(\Sigma_k^{\text{CFL}}) = \Delta_{k+1}^{\text{CFL}}$. Similarly, we obtain $\Pi_k^{\text{CFL}} \subseteq \text{DCFL}_T(\Pi_k^{\text{CFL}}) = \text{DCFL}_T(\Sigma_k^{\text{CFL}}) = \Delta_{k+1}^{\text{CFL}}$, where the first equality comes from (1). Moreover, since $\text{DCFL}_T^A \subseteq \text{CFL}_T^A$ for all oracles A , it follows that $\Delta_{k+1}^{\text{CFL}} = \text{DCFL}_T(\Sigma_k^{\text{CFL}}) \subseteq \text{CFL}_T(\Sigma_k^{\text{CFL}}) = \Sigma_{k+1}^{\text{CFL}}$. Finally, using the fact that $\text{DCFL}_T^A = \text{co-DCFL}_T^A$ for any oracle A , we easily obtain $\Delta_{k+1}^{\text{CFL}} = \text{co-}\Delta_{k+1}^{\text{CFL}} \subseteq \text{co-}\Sigma_{k+1}^{\text{CFL}} = \Pi_{k+1}^{\text{CFL}}$.

(3) Note that CFL belongs to $\text{DSPACE}(O(\log n))$ [25], which is obviously included in $\text{DSPACE}(O(n))$. Moreover, the fact that A is in $\text{DSPACE}(O(n))$ leads to $\text{CFL}_T^A \subseteq \text{DSPACE}(O(n))$, implying the desired containment $\Sigma_k^{\text{CFL}} \subseteq \text{DSPACE}(O(n))$ for every index $k \in \mathbb{N}^+$. \square

Hereafter, we will explore fundamental properties of our new hierarchy. Our starting point is a closure property under substitution. A *substitution* on alphabet Σ is actually a function $s : \Sigma \rightarrow \mathcal{P}(\Sigma^*)$ and this function is extended from the finite domain Σ into the infinite domain Σ^* as follows. Given a string $x = x_1 x_2 \dots x_n$, where each x_i is a symbol in Σ , we set $s(x)$ to be the language $\{y_1 y_2 \dots y_n \mid i \in [n], y_i \in s(x_i)\}$.

Moreover, for any language $A \subseteq \Sigma^*$, we define $s(A) = \bigcup_{x \in A} s(x)$. Each language family Σ_k^{CFL} is closed under substitution in the following sense.

Lemma 4.3 (*substitution property*) *Let $k \in \mathbb{N}^+$ and let s be any substitution on alphabet Σ satisfying $s(\sigma) \in \Sigma_k^{\text{CFL}}$ for each symbol $\sigma \in \Sigma$. For any language A over Σ , if L is in Σ_k^{CFL} , then $s(L)$ is also in Σ_k^{CFL} .*

Proof. Since the basis case $k = 1$ is well-known to hold (see, e.g., [7]), it suffices to assume that $k \geq 2$. For each symbol σ in Σ , take an oracle npda M_σ that recognizes $s(\sigma)$ relative to a certain language A_σ in $\Sigma_{k-1}^{\text{CFL}}$. In addition, let M denote an oracle npda recognizing L relative to a certain oracle $A \in \Sigma_{k-1}^{\text{CFL}}$. For simplicity, we assume that, when each oracle npda halts, it must empty its own stack (except for its bottom marker). Consider a new oracle npda N that behaves in the following manner. On input w , N nondeterministically splits w into (y_1, y_2, \dots, y_n) satisfying $w = y_1 y_2 \dots y_n$. Sequentially, at stage $i \in [n]$, N guesses a symbol, say, $\sigma_i \in \Sigma$ and simulates using a stack one or more steps of M while reading σ_i . In the end of this simulation stage, N places a special marker $\#$ on the top of the stack in order to share the same stack with M_{σ_i} . Next, N simulates M_{σ_i} on the input $\#y_i\$$ using an empty portion of the stack, provided that $\#$ is regarded a new bottom marker for M_{σ_i} . To make intact the saved data in the stack during this simulation of M_{σ_i} , whenever M_{σ_i} tries to remove $\#$, N instantly aborts the simulation and halts in a rejecting state. When M_{σ_i} queries a string, say, z_i , N instead produces $\sigma_i z_i$ on its query tape, where σ_i indicates which oracle is targeted. If all npda's M_{σ_i} enter certain accepting states, then N accepts w ; otherwise, it rejects w .

Finally, an oracle B is defined as the set of strings of the form σz for which z is in A_σ , where $\sigma \in \Sigma$. It can be observed that w is in $s(L)$ if and only if N accepts w relative to B . Moreover, since all A_σ 's are in $\Sigma_{k-1}^{\text{CFL}}$, the set B is also in $\Sigma_{k-1}^{\text{CFL}}$. Therefore, $s(L)$ belongs to $\text{CFL}_m^B \subseteq \Sigma_k^{\text{CFL}}$. \square

Once the closure property under substitution is established for Σ_k^{CFL} , other well-known closure properties (except for reversal and inverse homomorphism) follow directly.

Lemma 4.4 *For each index $k \in \mathbb{N}^+$, the family Σ_k^{CFL} is closed under the following operations: concatenation, union, reversal, Kleene closure, homomorphism, and inverse homomorphism.*

Proof. When $k = 1$, Σ_1^{CFL} (= CFL) satisfies all the listed closure properties (see, e.g., [7, 11]). Hereafter, we assume that $k \geq 2$. All the closure properties except for reversal and inverse homomorphism follow directly from Lemma 4.3. For completeness, however, we will include the proofs of those closure properties. The remaining closure properties require different arguments.

[union] Given two languages A_1 and A_2 in Σ_k^{CFL} , we define $L = \{1, 2\}$ and take a substitution s satisfying $s(i) = A_i$ for each $i \in \{1, 2\}$. Since $s(L) = A_1 \cup A_2$, Lemma 4.3 implies that $s(L)$ belongs to Σ_k^{CFL} .

[concatenation] For any languages $A_1, A_2 \in \Sigma_k^{\text{CFL}}$, we set $L = \{12\}$ and define $s(1) = A_1$ and $s(2) = A_2$. Since $s(L) = \{xy \mid x \in A_1, y \in A_2\}$, we apply Lemma 4.3 to $s(L)$ and obtain the desired containment $s(L) \in \Sigma_k^{\text{CFL}}$.

[Kleene closure] Given any language A in Σ_k^{CFL} , we define $s(1) = A$ and $L = \{1\}^*$, which obviously imply $s(L) = A^*$. Now, we apply Lemma 4.3 and then obtain the desired membership $s(L) \in \Sigma_k^{\text{CFL}}$.

[homomorphism] This is trivial since a homomorphism is a special case of a substitution.

[inverse homomorphism] Let Σ and Γ be two alphabets and take any language A in Σ_k^{CFL} over Γ and any homomorphism h from Σ to Γ^* . Our goal is to show that $h^{-1}(A)$ is in Σ_k^{CFL} . Let M be an oracle npda that recognizes A relative to an oracle, say, B in $\Sigma_{k-1}^{\text{CFL}}$. Now, we will construct another oracle npda N . Given any input $x = x_1 x_2 \dots x_n$ of length n , N applies h symbol by symbol. On reading x_i , N simulates several steps (including λ -moves) of M 's computation conducted during the scanning of $h(x_i)$. If N accepts x using B as an oracle, then the string $h(x) = h(x_1) \dots h(x_n)$ is in A ; otherwise, $h(x)$ is not in A . Thus, $h^{-1}(A)$ belongs to $\text{CFL}_T^B \subseteq \Sigma_k^{\text{CFL}}$.

[reversal] This proof proceeds by induction on $k \in \mathbb{N}^+$. As noted before, it suffices to show the induction step $k \geq 2$. Assume that $A \in \Sigma_k^{\text{CFL}}$ and let M be an oracle npda that recognizes A relative to a certain oracle $B \in \Sigma_{k-1}^{\text{CFL}}$. We aim at proving that the reversal $A^R = \{x^R \mid x \in A\}$ also belongs to Σ_k^{CFL} . Now, we will construct the desired reversing npda M_R . First, we conveniently set our new input instance is of the form $\$x^R\#$. Intuitively, we need to “reverse” the entire computation of M from an accepting configuration with the head scanning $\$$ to an initial configuration. To make the following description simple, we assume that M has only one accepting state and that M empties its stack before entering a halting state.

A major deviation from a standard proof of the case $k = 1$ is the presence of a query tape and a process of querying a word and receiving its oracle answer. Now, let us consider a situation that M produces a query word y and receives its oracle answer b . Since we try to reverse the entire computation of M , conceptually, we need to design M_R to (i) receive the oracle answer b from an oracle and then (ii) produce the reversed word y^R on the query tape. However, we cannot know the oracle answer before actually making a query. To avoid this pitfall, we force M_R to guess b and start producing y^R . After finishing y^R , we force M_R to make an actual query. If its actual oracle answer equals b , then M_R continues the simulation of M ; otherwise, M_R enters a rejecting state. To make this strategy work, we also need the reversed oracle B^R in place of B . By the induction hypothesis, the set B^R is in $\Sigma_{k-1}^{\text{CFL}}$. By formalizing the above argument, L^R can be shown to be recognized by M_R relative to B^R . \square

In Example 3.1, we have seen that the two languages Dup_2 and Dup_3 are in Σ_2^{CFL} . Since they are not context-free, these examples actually prove that $\Sigma_1^{\text{CFL}} \neq \Sigma_2^{\text{CFL}}$. Since $\text{co-CFL} \not\subseteq \text{CFL}/n$ [20] and $\text{co-CFL} \subseteq \Sigma_2^{\text{CFL}}$, we obtain a slightly improved separation as shown in Proposition 4.5.

Proposition 4.5 $\Sigma_2^{\text{CFL}} \not\subseteq \text{CFL}/n$.

Let us recall the language family BHCFL, the Boolean hierarchy over CFL. Here, we will show that the second level of the CFL hierarchy contains BHCFL.

Proposition 4.6 $\text{BHCFL} \subseteq \Sigma_2^{\text{CFL}} \cap \Pi_2^{\text{CFL}}$.

Proof. Obviously, $\text{CFL}_1 \subseteq \Sigma_2^{\text{CFL}}$ holds. It is therefore enough to show that $\text{CFL}_k \subseteq \Sigma_2^{\text{CFL}}$ for every index $k \geq 2$. For this purpose, we will present a simple characterization of the k th level of the Boolean hierarchy over CFL, despite the fact that $\text{CFL} \wedge \text{CFL} \neq \text{CFL}$.

Claim 8 For every index $k \geq 1$, $\text{CFL}_{2k} = \bigvee_{i \in [k]} \text{CFL}_2$ and $\text{CFL}_{2k+1} = (\bigvee_{i \in [k]} \text{CFL}_2) \vee \text{CFL}$.

Proof. In [24, Claim 4], it is shown that $\text{BCFL}_{2k} = \text{BCFL}_{2k-2} \vee \text{BCFL}_2$ (and thus $\text{BCFL}_{2k} = \bigvee_{i \in [k]} \text{BCFL}_2$ follows) for the family BCFL of *bounded context-free languages*. Essentially the same proof works to show that $\text{CFL}_{2k} = \bigvee_{i \in [k]} \text{CFL}_2$. Moreover, since $\text{CFL}_{2k+1} = \text{CFL}_{2k} \vee \text{CFL}$ by the definition, we obtain $\text{CFL}_{2k+1} = (\bigvee_{i \in [k]} \text{CFL}_2) \vee \text{CFL}$. \square

Now, we want to show that $\text{CFL}_{2k}, \text{CFL}_{2k+1} \subseteq \Sigma_2^{\text{CFL}}$ for all indices $k \geq 1$. We proceed by induction on $k \geq 1$. The case of $k = 2$ will be shown as follows.

Claim 9 $\text{CFL}_2 \subseteq \Sigma_2^{\text{CFL}}$.

Proof. Let L be any language in CFL_2 and take two context-free languages A and B satisfying $L = A \cap \overline{B}$. Let M be an appropriate npda recognizing A . Consider the following procedure: on input x , copy x to the query tape and, at the same time, run M on x . When M enters an accepting state along a certain computation path, make a query x to \overline{B} . This demonstrates that L is in $\text{CFL}_m^{\overline{B}}$, which is included in $\text{CFL}_m^{\text{co-CFL}} \subseteq \text{CFL}_T^{\text{co-CFL}} = \Sigma_2^{\text{CFL}}$. \square

Assuming $k \geq 2$, let us consider the language family CFL_{2k} . Claim 8 implies that $\text{CFL}_{2k} = \bigvee_{i \in [k]} \text{CFL}_2$. Since $\text{CFL}_2 \subseteq \Sigma_2^{\text{CFL}}$ by Claim 9, we obtain $\text{CFL}_{2k} \subseteq \bigvee_{i \in [k]} \Sigma_2^{\text{CFL}}$. As is shown in Lemma 4.9, Σ_2^{CFL} is closed under union and this fact implies that $\text{CFL}_{2k} \subseteq \Sigma_2^{\text{CFL}}$. Next, we consider CFL_{2k+1} . Since $\text{CFL}_{2k+1} = \text{CFL}_{2k} \vee \text{CFL}$ by the definition, the above argument implies that $\text{CFL}_{2k+1} \subseteq \Sigma_2^{\text{CFL}} \vee \text{CFL}$. Since $\text{CFL} \subseteq \Sigma_2^{\text{CFL}}$ and the closure property of Σ_2^{CFL} under union, it follows that $\text{CFL}_{2k+1} \subseteq \Sigma_2^{\text{CFL}} \vee \Sigma_2^{\text{CFL}} = \Sigma_2^{\text{CFL}}$. As a consequence, we conclude that $\text{CFL}_{2k}, \text{CFL}_{2k+1} \subseteq \Sigma_2^{\text{CFL}}$. Therefore, $\text{BHCFL} \subseteq \Sigma_2^{\text{CFL}}$ holds.

Furthermore, we will prove that $\text{BHCFL} \subseteq \Pi_2^{\text{CFL}}$. It is possible to prove by induction on $k \in \mathbb{N}^+$ that $\text{co-CFL}_k \subseteq \text{CFL}_{k+1}$. From this inclusion, we obtain $\text{co-BHCFL} \subseteq \text{BHCFL}$. By symmetry, $\text{BHCFL} \subseteq \text{co-BHCFL}$ holds. Thus, we conclude that $\text{BHCFL} = \text{co-BHCFL}$. Therefore, the earlier assertion $\text{BHCFL} \subseteq \Sigma_2^{\text{CFL}}$ implies $\text{BHCFL} \subseteq \Pi_2^{\text{CFL}}$ as well. \square

Let us turn our attention to $\text{CFL}(\omega)$. A direct estimation of each language family $\text{CFL}(k)$ shows that $\text{CFL}(\omega)$ is included in BHCFL.

Proposition 4.7 $\text{CFL}(\omega) \subseteq \text{BHCFL}$ (thus, $\text{CFL}(\omega) \subseteq \Sigma_2^{\text{CFL}} \cap \Pi_2^{\text{CFL}}$).

Proof. A key to the proof of the first part of this proposition is the following claim.

Claim 10 For every index $k \geq 1$, $\text{CFL}(k) \subseteq \text{CFL}_{2k+1}$ holds.

Proof. We will proceed by induction on $k \geq 1$. When $k = 1$, the claim is obviously true since $\text{CFL}(1) = \text{CFL}_1 \subseteq \text{CFL}_3$. For the induction step, assume that $k \geq 2$. The induction hypothesis implies that $\text{CFL}(k-1) \subseteq \text{CFL}_{2k-1}$. Since $\text{CFL}(k) = \text{CFL}(k-1) \wedge \text{CFL}$, we obtain $\text{CFL}(k) \subseteq \text{CFL}_{2k-1} \wedge \text{CFL}$. In contrast, it follows by the definition that $\text{CFL}_{2k+1} = \text{CFL}_{2k} \vee \text{CFL} = (\text{CFL}_{2k-1} \wedge \text{co-CFL}) \vee \text{CFL}$. The last term equals $(\text{CFL}_{2k-1} \vee \text{CFL}) \wedge (\text{CFL} \vee \text{co-CFL})$. Clearly, this language family includes $\text{CFL}_{2k-1} \wedge \text{CFL}$ as a subclass. Therefore, we conclude that $\text{CFL}(k) \subseteq \text{CFL}_{2k+1}$. \square

By Claim 10, it follows that $\text{CFL}(\omega) = \bigcup_{k \in \mathbb{N}^+} \text{CFL}(k) \subseteq \bigcup_{k \in \mathbb{N}^+} \text{CFL}_{2k+1} \subseteq \text{BHCFL}$. The second part of the proposition follows from Proposition 4.6 \square

Let us argue that the language family $\text{CFL}_m^{\text{CFL}(\omega)}$ is located within the third level of the CFL hierarchy.

Proposition 4.8 $\text{CFL}_m^{\text{CFL}(\omega)} \subseteq \Sigma_3^{\text{CFL}}$.

Proof. Proposition 4.7 implies that $\text{CFL}_m^{\text{CFL}(\omega)}$ is included in $\text{CFL}_m^{\text{BHCFL}}$. By Theorem 4.6, it follows that $\text{CFL}_m^{\text{BHCFL}}$ is included in $\text{CFL}_m(\Pi_2^{\text{CFL}})$, which is obviously a subclass of $\text{CFL}_T(\Pi_2^{\text{CFL}}) = \Sigma_3^{\text{CFL}}$. \square

4.2 Structural Properties of the CFL Hierarchy

After showing fundamental properties of languages in the CFL hierarchy in Section 4.1, we will further explore structural properties that characterize the CFL hierarchy. Moreover, we will present three alternative characterizations (Theorem 4.11 and Proposition 4.15) of the hierarchy.

Let us consider a situation in which Boolean operations are applied to languages in the CFL hierarchy.

Lemma 4.9 1. $\Sigma_k^{\text{CFL}} \vee \Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$ and $\Pi_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ for any $k \geq 1$.
2. $\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}} \subseteq \Sigma_{k+1}^{\text{CFL}} \cap \Pi_{k+1}^{\text{CFL}}$ and $\Sigma_k^{\text{CFL}} \vee \Pi_k^{\text{CFL}} \subseteq \Sigma_{k+1}^{\text{CFL}} \cap \Pi_{k+1}^{\text{CFL}}$ for any $k \geq 1$.

Proof. In what follows, we are focused only on the Σ_k^{CFL} case since the Π_k^{CFL} case is symmetric.

(1) When $k = 1$, since CFL is closed under union, the statement follows immediately. Assume that $k \geq 2$ and let L be any language in $\Sigma_k^{\text{CFL}} \vee \Sigma_k^{\text{CFL}}$. Now, we take two oracle npda's M_0, M_1 and two languages $A, B \in \Pi_{k-1}^{\text{CFL}}$ satisfying that $L = L(M_0, A) \cup L(M_1, B)$. Our goal is to show that $L \in \Sigma_k^{\text{CFL}}$. Let us consider another oracle npda M that behaves as follows. On input x , M guesses a bit b , writes it down on a query tape, and simulates M_b on x . Thus, when M_b halts with a query word y_b produced on its query tape, N does the same with by_b . Let us define C as the union $\{0y \mid y \in A\} \cup \{1y \mid y \in B\}$. We will show that C is in Π_{k-1}^{CFL} . To see this fact, consider the complement \overline{C} . Note that $\overline{C} = \{0y \mid y \in \overline{A}\} \cup \{1y \mid y \in \overline{B}\}$. Because $\overline{A}, \overline{B} \in \Sigma_{k-1}^{\text{CFL}}$, by induction hypothesis, \overline{C} belongs to $\Sigma_{k-1}^{\text{CFL}}$. Since $L = L(N, C)$ holds, we conclude that $L \in \text{CFL}_T^C \subseteq \Sigma_k^{\text{CFL}}$.

(2) Assuming $k \geq 2$, let L be any language in $\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}$ and take an oracle npda M and two languages $A \in \Pi_{k-1}^{\text{CFL}}$ and $B \in \Pi_k^{\text{CFL}}$ for which $L = L(M, A) \cap B$. Here, we define a new oracle npda N to simulate M on input x and produce an encoding $[\frac{\tilde{x}}{y}]$ of a computation of M on x with query word y . Next, let us define C as the set $\{[\frac{\tilde{x}}{y}] \mid x \in A, y \in B\}$, which belongs to Π_k^{CFL} by (1) using a fact that $\Pi_{k-1}^{\text{CFL}} \subseteq \Pi_k^{\text{CFL}}$. Since $L = L(N, C)$, L is in CFL_T^C , which is a subclass of $\Sigma_{k+1}^{\text{CFL}}$. In a similar fashion, we can prove that $L \in \Pi_{k+1}^{\text{CFL}}$. \square

What is missing in the list of the above lemma is two language families $\Sigma_k^{\text{CFL}} \wedge \Sigma_k^{\text{CFL}}$ and $\Pi_k^{\text{CFL}} \vee \Pi_k^{\text{CFL}}$. As we have seen, it holds that $\text{CFL} \wedge \text{CFL} = \text{CFL}(2) \neq \text{CFL}$. Therefore, the equality $\Sigma_k^{\text{CFL}} \wedge \Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$ does not hold in the first level (i.e., $k = 1$). Surprisingly, it is possible to prove that this equality actually holds for any level *more than* 1.

Proposition 4.10 $\Sigma_k^{\text{CFL}} \wedge \Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$ and $\Pi_k^{\text{CFL}} \vee \Pi_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ for all levels $k \geq 2$.

This proposition is not quite trivial and its proof requires two new characterizations of Σ_k^{CFL} in terms of many-one reducibilities. Note that these characterizations are a natural extension of Claim 2 and, for our purpose, we introduce two many-one hierarchies. The *many-one CFL hierarchy* consists of language families $\Sigma_{m,k}^{\text{CFL}}$ and $\Pi_{m,k}^{\text{CFL}}$ ($k \in \mathbb{N}$) defined as follows: $\Sigma_{m,0}^{\text{CFL}} = \Pi_{m,0}^{\text{CFL}} = \text{DCFL}$, $\Sigma_{m,1}^{\text{CFL}} = \text{CFL}$, $\Pi_{m,k}^{\text{CFL}} = \text{co-}\Sigma_{m,k}^{\text{CFL}}$,

and $\Sigma_{m,k+1}^{\text{CFL}} = \text{CFL}_m(\Pi_{m,k}^{\text{CFL}})$ for any $k \geq 1$, where the subscript “ m ” stands for “many-one.” A *relativized many-one NFA hierarchy*, which was essentially formulated in [14], is defined as follows relative to oracle A : $\Sigma_{m,0}^{\text{NFA},A} = \text{DFA}_m^A$, $\Sigma_{m,1}^{\text{NFA},A} = \text{NFA}_m^A$, $\Pi_{m,k}^{\text{NFA},A} = \text{co-}\Sigma_{m,k}^{\text{NFA},A}$, and $\Sigma_{m,k+1}^{\text{NFA},A} = \text{NFA}_m(\Pi_{m,k}^{\text{NFA},A})$ for every index $k \geq 1$. Given a language family \mathcal{C} , the notation $\Sigma_{m,k}^{\text{NFA},\mathcal{C}}$ (or $\Sigma_{m,k}^{\text{NFA}}(\mathcal{C})$) denotes the union $\bigcup_{A \in \mathcal{C}} \Sigma_{m,k}^{\text{NFA},A}$.

Theorem 4.11 $\Sigma_k^{\text{CFL}} = \Sigma_{m,k}^{\text{CFL}} = \Sigma_{m,k}^{\text{NFA}}(\text{DYCK})$ for every index $k \geq 1$.

Since the proof of Theorem 4.11 is involved, prior to the proof, we will demonstrate how to prove Proposition 4.10 using the theorem.

Proof of Proposition 4.10. In what follows, it suffices to prove that $\Sigma_k^{\text{CFL}} \wedge \Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$, since $\Pi_k^{\text{CFL}} \vee \Pi_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ is obtained by symmetry. First, take any language L in $\Sigma_k^{\text{CFL}} \wedge \Sigma_k^{\text{CFL}}$ and assume that $L = L_1 \cap L_2$ for two languages $L_1, L_2 \in \Sigma_k^{\text{CFL}}$. Theorem 4.11 implies that L_1 and L_2 are both in $\Sigma_{m,k}^{\text{NFA}}(\text{DYCK})$. Now, we choose oracle npda’s M_1 and M_2 that respectively recognize L_1 and L_2 relative to oracles A_1 and A_2 , where $A_1, A_2 \in \Pi_{m,k-1}^{\text{NFA}}(\text{DYCK})$. Let us consider a new npda N that works in the following fashion. In scanning each input symbol, say, σ , N simulates in parallel one or more steps of M_1 and M_2 using two sets of inner states for M_1 and M_2 . Such a parallel simulation of two machines is possible because M_1 and M_2 use no stacks. Moreover, whenever M_1 (resp., M_2) tries to write a symbol, N writes it on the upper (resp., lower) track of its query tape. To write two query strings y_1 and y_2 of M_1 and M_2 , respectively, onto N ’s single query tape, we actually write their \natural -extensions. Now, let $[\frac{y}{z}]$ denote a query string produced by N so that $[\frac{y}{z}]$ encodes computations of M_1 and M_2 . A new oracle B is finally set to be $\{[\frac{y}{z}] \mid y \in A_1, z \in A_2\}$. Since $\Pi_{k-1}^{\text{CFL}} = \Pi_{m,k-1}^{\text{NFA}}(\text{DYCK})$, Lemma 4.9(1) ensures that B is also in $\Pi_{m,k-1}^{\text{NFA}}(\text{DYCK})$. By the above definitions, it holds that N m -reduces L to B . Therefore, it immediately follows that $L \in \text{NFA}_m^B \subseteq \Sigma_{m,k}^{\text{NFA}}(\text{DYCK}) = \Sigma_k^{\text{CFL}}$. \square

The first step toward the proof of Theorem 4.11 is to prove a key lemma given below.

Lemma 4.12 For every index $k \geq 1$, it holds that $\Sigma_{k+1}^{\text{CFL}} \subseteq \text{CFL}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}) \subseteq \text{NFA}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}})$.

Proof. The proof of the lemma proceeds by induction on $k \in \mathbb{N}^+$. Notice that the base case $k = 1$ has been already proven as Proposition 3.13. Therefore, in what follows, we aim at the induction step of $k \geq 2$ by proving the following two inclusions: (1) $\Sigma_{k+1}^{\text{CFL}} \subseteq \text{CFL}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}})$ and (2) $\text{CFL}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}) \subseteq \text{NFA}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}})$.

(1) Let us recall the proof of Proposition 3.13, in particular, the proof of the following inclusion: $\text{CFL}_T^{\text{CFL}} \subseteq \text{CFL}_m(\text{CFL} \wedge \text{co-CFL})$. We note that this proof is *relativizable* (that is, it works when we append an oracle to underlying npda’s). To be more precise, essentially the same proof proves that $\text{CFL}_T(\text{CFL}_T^A) \subseteq \text{CFL}_m(\text{CFL}_T^A \wedge \text{co-CFL}_T^A)$ for any oracle A . If we choose an arbitrary language in $\Sigma_{k-1}^{\text{CFL}}$ as A , then we conclude that $\Sigma_{k+1}^{\text{CFL}} \subseteq \text{CFL}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}})$.

(2) By setting $\mathcal{C} = \Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}$ in Lemma 3.5, we obtain $\text{CFL}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}) \subseteq \text{NFA}_m(\text{DCFL} \wedge \mathcal{C})$. Note that $\text{DCFL} \wedge (\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}) = \Sigma_k^{\text{CFL}} \wedge (\text{DCFL} \wedge \Pi_k^{\text{CFL}})$. Since $\text{DCFL} \subseteq \Pi_k^{\text{CFL}}$, it instantly follows that $\text{DCFL} \wedge \Pi_k^{\text{CFL}} \subseteq \Pi_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ by Lemma 4.9(1). In summary, we obtain the desired inclusion $\text{NFA}_m(\text{DCFL} \wedge \mathcal{C}) \subseteq \text{NFA}_m(\Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}})$. \square

The second step is to establish the following inclusion relationship between two language families $\text{NFA}_m(\Sigma_{m,k}^{\text{CFL}} \wedge \Pi_{m,e}^{\text{CFL}})$ and $\text{CFL}_m(\Pi_{m,e}^{\text{CFL}})$.

Lemma 4.13 For any two indices $k \geq 1$ and $e \geq k - 1$, it holds that $\text{NFA}_m(\Sigma_{m,k}^{\text{CFL}} \wedge \Pi_{m,e}^{\text{CFL}}) \subseteq \text{CFL}_m(\Pi_{m,e}^{\text{CFL}})$.

Proof. Let L be any language in NFA_m^A , where A is a certain language in $\Sigma_{m,k}^{\text{CFL}} \wedge \Pi_{m,e}^{\text{CFL}}$. Now, we express A as $A_1 \cap A_2$ using appropriate languages $A_1 \in \text{CFL}_m^{B_1}$ and $A_2 \in \Pi_{m,e}^{\text{CFL}}$, where $B_1 \in \Pi_{m,k-1}^{\text{CFL}}$. Our goal is to construct an oracle npda N and an oracle C for L . The desired machine N takes input x and simulates M on x . When M tries to write a symbol, say, σ , N writes σ on the upper track of its query tape and also simulates one or more steps of M_1 ’s computation during the scanning of σ using a stack. When M_1 writes a symbol, N uses the lower track to keep the symbol. Finally, N produces a query string of the form $[\frac{y}{z}]$, where y is a query word of M and z is a query word of M_1 . Next, we define C to be $\{[\frac{y}{z}] \mid y \in A_2, z \in B_1\}$. This language C obviously belongs to $\Pi_{m,e}^{\text{CFL}} \wedge \Pi_{m,k-1}^{\text{CFL}}$, which equals $\Pi_{m,e}^{\text{CFL}}$ by Lemma 4.9(1), since $e \geq k - 1$.

Therefore, L is in $\text{CFL}_m^C \subseteq \text{CFL}_m(\Pi_{m,e}^{\text{CFL}})$. \square

Finally, we are ready to give the proof of Theorem 4.11.

Proof of Theorem 4.11. The proof of the theorem proceeds by induction on $k \geq 1$. Since Lemma 3.3 handles the base case $k = 1$, it is sufficient to assume that $k \geq 2$. First, we will show the second equality given in the theorem.

Claim 11 For any index $k \geq 1$, $\Sigma_{m,k}^{\text{CFL}} = \Sigma_{m,k}^{\text{NFA}}(\text{DYCK})$ holds.

Proof. If $k = 1$, then the claim is exactly the same as Claim 2. In the case of $k \geq 2$, assume that $L \in \text{CFL}_m^A$ for a certain language A in $\Pi_{m,k-1}^{\text{CFL}}$. A proof similar to that of Claim 2 proves the existence of a certain Dyck language D satisfying that $\text{CFL}_m^A = \text{NFA}_m^B$, where B is of the form $\{[\frac{\tilde{y}}{\tilde{z}}] \mid y \in D, z \in A\}$ and \tilde{y} and \tilde{z} are \natural -extensions of y and z , respectively. The definition places B into the language family $\text{DCFL} \wedge \Pi_{m,k-1}^{\text{CFL}}$, which equals $\Pi_{m,k-1}^{\text{CFL}}$ because of $k \geq 2$. By the induction hypothesis, $\Pi_{m,k-1}^{\text{CFL}} = \Pi_{m,k-1}^{\text{NFA}}(\text{DYCK})$ holds. It thus follows that $\text{NFA}_m^B \subseteq \text{NFA}_m(\Pi_{m,k-1}^{\text{NFA}}(\text{DYCK})) = \Sigma_{m,k}^{\text{NFA}}(\text{DYCK})$, and thus we obtain $L \in \Sigma_{m,k}^{\text{NFA}}(\text{DYCK})$. \square

Next, we will establish the first equality in the theorem. Clearly, $\Sigma_{m,k}^{\text{CFL}} \subseteq \Sigma_k^{\text{CFL}}$ holds since $\text{CFL}_m^A \subseteq \text{CFL}_T^A$ for any oracle A . Now, we target the other inclusion. By Lemma 4.12, it follows that $\Sigma_k^{\text{CFL}} \subseteq \text{NFA}_m(\Sigma_{k-1}^{\text{CFL}} \wedge \Pi_{k-1}^{\text{CFL}})$. Since $\Sigma_{k-1}^{\text{CFL}} = \Sigma_{m,k-1}^{\text{CFL}}$, we obtain $\Sigma_k^{\text{CFL}} \subseteq \text{NFA}_m(\Sigma_{m,k-1}^{\text{CFL}} \wedge \Pi_{m,k-1}^{\text{CFL}})$. Lemma 4.13 further implies that $\text{NFA}_m(\Sigma_{m,k-1}^{\text{CFL}} \wedge \Pi_{m,k-1}^{\text{CFL}}) \subseteq \text{CFL}_m(\Pi_{m,k-1}^{\text{CFL}}) = \Sigma_{m,k}^{\text{CFL}}$. In conclusion, $\Sigma_k^{\text{CFL}} \subseteq \Sigma_{m,k}^{\text{CFL}}$ holds. \square

An *upward collapse property* holds for the CFL hierarchy except for the first level. Similar to the notation CFL_e expressing the e th level of the Boolean hierarchy over CFL, a new notation $\Sigma_{k,e}^{\text{CFL}}$ is introduced to denote the e th level of the Boolean hierarchy over Σ_k^{CFL} . Additionally, we set $\text{BH}\Sigma_k^{\text{CFL}} = \bigcup_{e \in \mathbb{N}^+} \Sigma_{k,e}^{\text{CFL}}$. Notice that, when $k = 1$, $\text{BH}\Sigma_1^{\text{CFL}}$ coincides with BHCFL .

Lemma 4.14 (*upward collapse properties*) Let k be any integer at least 2.

1. $\Sigma_k^{\text{CFL}} = \Sigma_{k+1}^{\text{CFL}}$ if and only if $\text{CFLH} = \Sigma_k^{\text{CFL}}$.
2. $\Sigma_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ if and only if $\text{BH}\Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$.
3. $\Sigma_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ implies $\Sigma_k^{\text{CFL}} = \Sigma_{k+1}^{\text{CFL}}$.

Proof. (1) It is obvious that $\text{CFLH} = \Sigma_k^{\text{CFL}}$ implies $\Sigma_k^{\text{CFL}} = \Sigma_{k+1}^{\text{CFL}}$. Now, assume that $\Sigma_k^{\text{CFL}} = \Sigma_{k+1}^{\text{CFL}}$. By applying the complementation operation, we obtain $\Pi_k^{\text{CFL}} = \Pi_{k+1}^{\text{CFL}}$. Thus, it follows that $\Sigma_{k+2}^{\text{CFL}} = \text{CFL}_T(\Pi_{k+1}^{\text{CFL}}) = \text{CFL}_T(\Pi_k^{\text{CFL}}) = \Sigma_{k+1}^{\text{CFL}}$. Similarly, it is possible to prove by induction on $e \in \mathbb{N}^+$ that $\Sigma_{k+e}^{\text{CFL}} = \Sigma_k^{\text{CFL}}$. Therefore, $\text{CFLH} = \Sigma_k^{\text{CFL}}$ holds.

(2) Since $\Pi_k^{\text{CFL}} \subseteq \text{BH}\Sigma_k^{\text{CFL}}$, obviously $\text{BH}\Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$ implies $\Sigma_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$. Next, assume that $\Sigma_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$. By induction on $e \in \mathbb{N}^+$, we wish to prove that $\Sigma_{k,e}^{\text{CFL}} \subseteq \Sigma_k^{\text{CFL}}$. The case where $e = 1$ is trivial. Firstly, let us consider the language family $\Sigma_{k,2e+1}^{\text{CFL}}$ for $e \geq 1$. Note that the induction hypothesis implies $\Sigma_{k,2e}^{\text{CFL}} \subseteq \Sigma_k^{\text{CFL}}$. It thus holds that $\Sigma_{k,2e+1}^{\text{CFL}} = \Sigma_{k,2e}^{\text{CFL}} \vee \Sigma_k^{\text{CFL}} \subseteq \Sigma_k^{\text{CFL}} \vee \Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$ by Lemma 4.9(1). Secondly, we consider the family $\Sigma_{k,2e+2}^{\text{CFL}}$. It holds that $\Sigma_{k,2e+2}^{\text{CFL}} = \Sigma_{k,2e+1}^{\text{CFL}} \wedge \Pi_k^{\text{CFL}} \subseteq \Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}}$. Since $\Pi_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$, we obtain $\Sigma_{k,2e+2}^{\text{CFL}} \subseteq \Sigma_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}} = \Pi_k^{\text{CFL}} \wedge \Pi_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ by Lemma 4.9(1). The last term obviously equals Σ_k^{CFL} from our assumption. Overall, we conclude that $\text{BH}\Sigma_k^{\text{CFL}} = \Sigma_k^{\text{CFL}}$.

(3) Assume that $\Sigma_k^{\text{CFL}} = \Pi_k^{\text{CFL}}$ and focus our attention to $\Sigma_{k+1}^{\text{CFL}}$. Since Theorem 4.11 implies $\Sigma_k^{\text{CFL}} = \Sigma_{m,k}^{\text{CFL}}$, our assumption is equivalent to $\Sigma_{m,k}^{\text{CFL}} = \Pi_{m,k}^{\text{CFL}}$. By Lemma 4.12, it follows that $\Sigma_{k+1}^{\text{CFL}} \subseteq \text{NFA}_m(\Sigma_{m,k}^{\text{CFL}} \wedge \Pi_{m,k}^{\text{CFL}}) = \text{NFA}_m(\Sigma_{m,k}^{\text{CFL}} \wedge \Sigma_{m,k}^{\text{CFL}})$, which is included in $\text{NFA}_m(\Sigma_{m,k}^{\text{CFL}})$ by Lemma 4.9(1). Since $\Sigma_k^{\text{CFL}} \subseteq \Sigma_{m,k}^{\text{CFL}} \wedge \Pi_{m,k-1}^{\text{CFL}}$, Lemma 4.13 implies that $\text{NFA}_m(\Sigma_{m,k}^{\text{CFL}}) \subseteq \text{NFA}_m(\Sigma_{m,k}^{\text{CFL}} \wedge \Pi_{m,k-1}^{\text{CFL}}) \subseteq \text{CFL}_m(\Pi_{m,k-1}^{\text{CFL}}) = \Sigma_{m,k}^{\text{CFL}}$, which equals Σ_k^{CFL} by Theorem 4.11 again. \square

From Lemma 4.14, if the Boolean hierarchy over Σ_k^{CFL} collapses to Σ_k^{CFL} , then the CFL hierarchy collapses. It is not clear, however, that a much weaker assumption like $\Sigma_{k,e}^{\text{CFL}} = \Sigma_{k,e+1}^{\text{CFL}}$ suffices to draw the collapse of the CFL hierarchy (for instance, $\Sigma_{k+1}^{\text{CFL}} = \Sigma_{k+2}^{\text{CFL}}$).

We note that Theorem 4.11 also gives a logical characterization of Σ_k^{CFL} . We define a function Ext as $\text{Ext}(\tilde{x}) = x$ for any \natural -extension \tilde{x} of string x .

Proposition 4.15 Let $k \geq 1$. For any language $L \in \Sigma_k^{\text{CFL}}$ over alphabet Σ , there exists another language $A \in \text{DCFL}$ and a linear polynomial p with $p(n) \geq n$ for all $n \in \mathbb{N}$ that satisfy the following equivalence relation: for any number $n \in \mathbb{N}$ and any string $x \in \Sigma^n$,

$$x \in L \quad \text{if and only if} \quad \exists \tilde{x} (|\tilde{x}| \leq p(n)) \exists y_1 (|y_1| \leq p(n)) \forall y_2 (|y_2| \leq p(n)) \dots Q_k y_k (|y_k| \leq p(n)) [x = \text{Ext}(\tilde{x}) \wedge [\tilde{x}, y_1, y_2, \dots, y_k]^T \in A],$$

where Q_k is \exists if k is odd and is \forall if k is even. Moreover, \tilde{x} is a \natural -extension of x .

Let us recall from Section 3.2 the notion of “encoding of a computation path” of a nondeterministic computation.

Proof of Proposition 4.15. This proof proceeds by induction on $k \in \mathbb{N}^+$. First, we will target the case of $k = 1$ and we begin with the assumption that $L \in \Sigma_1^{\text{CFL}}$ ($= \text{CFL}$). Moreover, we assume that L is recognized by a certain npda, say, M and let p be a linear polynomial that bounds the running time of M . Consider the following new language A . This language A is defined as the collection of all strings of the form $[\frac{\tilde{x}}{y_1}]$ that encodes an accepting computation path of M on input x . It is not difficult to verify that $A \in \text{DCFL}$. Moreover, the definition of A indicates that, for every string x , x is in L if and only if there exist a \natural -extension \tilde{x} of x and $[\frac{\tilde{x}}{y_1}]$ is in A . The latter condition is logically equivalent to $\exists \tilde{x} (|\tilde{x}| \leq p(n)) \exists y_1 (|y_1| \leq p(n)) [x = \text{Ext}(\tilde{x}) \wedge [\frac{\tilde{x}}{y_1}] \in A]$.

For the induction step $k \geq 2$, let us assume that $L \in \Sigma_k^{\text{CFL}}$. Theorem 4.11 implies that L is many-one NFA-reducible to a certain oracle B in $\Pi_{m,k-1}^{\text{NFA}}(\text{DYCK}) (= \Pi_{k-1}^{\text{CFL}})$ via an oracle nfa M . Note that the running time of M is upper-bounded by a certain linear polynomial, say, p . Since \overline{B} is in $\Sigma_{k-1}^{\text{CFL}}$, our induction hypothesis ensures that there are a linear polynomial q and a language C in DCFL such that, for any string z_1 , z_1 is in \overline{B} if and only if $\exists \tilde{z}'_1 (|\tilde{z}'_1| \leq q(n')) \exists u_2 (|u_2| \leq q(n')) \dots Q'_k u_k (|u_k| \leq q(n')) [[z_1 = \text{Ext}(\tilde{z}'_1) \wedge [\tilde{z}'_1, u_2, \dots, u_k]^T \in C]]$, where Q'_k is \exists if k is even and is \forall if k is odd, and $n' = |z_1|$. In a similar way as in the base case of $k = 1$, we can define a language D in DCFL that is composed of strings $[\tilde{x}, u_1, \tilde{z}_1]^T$ that encodes an accepting computation path of M with input x and query word z_1 . Note that, for any given pair (\tilde{x}, u_1) , there is at most one string \tilde{z}_1 such that $[\tilde{x}, u_1, \tilde{z}_1]^T \in D$. From such a unique \tilde{z}_1 , a string $z_1 = \text{Ext}(\tilde{z}_1)$ is also uniquely determined. Note that x is in L if and only if there exist \natural -extensions \tilde{x} of input x and \tilde{z}'_1 of query word z_1 and u_1 is an accepting computation path such that $[\tilde{x}, u_1, \tilde{z}_1]^T \in D \wedge z_1 \in \overline{B}$.

To complete the proof, we want to combine two strings $[\tilde{x}, u_1, \tilde{z}_1]^T$ and $[\tilde{z}'_1, u_2, \dots, u_k]^T$ satisfying $\text{Ext}(\tilde{z}_1) = \text{Ext}(\tilde{z}'_1)$ into a single string by applying a technique of inserting \natural so that a single tape head can read off all information from the string at once. For convenience, we introduce three languages. Let $D' = \{[\frac{\tilde{x}}{\tilde{z}'_1}] \mid \exists u_1, \tilde{z}_1 [\text{Ext}(\tilde{w}_1) = [\frac{u_1}{\tilde{z}_1}] \wedge [\tilde{x}, u_1, \tilde{z}_1]^T \in D]\}$, $C' = \{[\tilde{w}_2, y_3, \dots, y_k]^T \mid \exists \tilde{z}'_1, y_2, \dots, y_k [\text{Ext}(\tilde{w}_2) = [\frac{\tilde{z}'_1}{y_2}] \wedge (\bigwedge_{i=3}^k \text{Ext}(y_i) = u_i) \wedge [\tilde{z}'_1, u_2, \dots, u_k]^T \in C]\}$, and $E = \{[\frac{\tilde{w}_1}{\tilde{z}'_1}] \mid \exists u_1, \tilde{z}_1 [\text{Ext}(\tilde{w}_1) = [\frac{u_1}{\tilde{z}_1}] \wedge \text{Ext}(\tilde{z}_1) = \text{Ext}(\tilde{z}'_1)]\}$. Finally, we define a language $G = \{[\tilde{x}, y_1, y_2, \dots, y_k]^T \mid [\frac{\tilde{x}}{y_1}] \in D' \wedge [\frac{y_1}{y_2}] \in E \wedge [y_1, y_2, \dots, y_k]^T \notin C'\}$. It is not difficult to show that G is in DCFL. Now, let $r(n) = q(p(n))$ for all $n \in \mathbb{N}$. With this language G , it follows that, for any string x , x is in L if and only if $\exists \tilde{x} (|\tilde{x}| \leq r(n)) \exists y_1 (|y_1| \leq p(n)) \forall y_2 (|y_2| \leq r(n)) \dots Q_k y_k (|y_k| \leq r(n)) [x = \text{Ext}(\tilde{x}) \wedge [\tilde{x}, y_1, y_2, \dots, y_k]^T \in G]$. Therefore, we have completed the induction step. \square

4.3 BPCFL and a Relativized CFL Hierarchy

Let us consider a probabilistic analogue of CFL. The *bounded-error probabilistic language family* BPCFL consists of all languages that are recognized by ppda's whose error probability is bounded from above by an absolute constant $\varepsilon \in [0, 1/2)$. This family is naturally contained in the *unbounded-error probabilistic language family* PCFL. Hromkovič and Schnitger [8] studied properties of BPCFL and showed that BPCFL and CFL are incomparable; more accurately, $\text{BPCFL} \not\subseteq \text{CFL}$ and $\text{CFL} \not\subseteq \text{BPCFL}$. It is possible to strengthen the last separation using advice in the following way.

Proposition 4.16 $\text{BPCFL} \not\subseteq \text{CFL}/n$.

Proof. Let us consider the example language Equal_6 that is composed of all strings w over the alphabet $\Sigma_6 = \{a_1, a_2, \dots, a_6, \#\}$ such that each symbol except $\#$ appears in w the same number of times. It is shown in [20] that Equal_6 does not belong to CFL/n . To complete this proof, it is enough to show that Equal_6 actually falls into BPCFL.

We set $N = 5$ and consider the following probabilistic procedure. Let w be any input and define $\alpha_i = \#_{a_i}(w)$ for every $i \in [6]$. In the case where all α_i 's are at most N , we deterministically determine whether $w \in \text{Equal}_6$ without using any stack. Hereafter, we consider the case where $\alpha_i > N$ for all $i \in [6]$. We randomly pick up two numbers x and y from $[N]$. We scan w from left to right. Whenever we scan a_1 (resp., a_2 and a_3), we push 1 (resp., 1^x and 1^y) into the stack. On the contrary, when we scan a_4 (resp., a_5 and a_6), we pop 1 (resp., 1^x and 1^y) from the stack. If the stack becomes empty during the pop-ups, then we instead push a special symbol “-1” to indicate that there is a deficit in the stack content. If this happens, when we push 1's, we actually pops the same number of -1's. After reading up w , if the stack becomes empty, then we accept the input; otherwise, we reject it. Note that there is $\ell = |(\alpha_1 - \alpha_4) + x(\alpha_2 - \alpha_5) + y(\alpha_3 - \alpha_6)|$ symbols in the stack.

If w is in Equal_6 , then $\ell = 0$ obviously holds for any choice of $(x, y) \in [N] \times [N]$. Conversely, we assume that $w \notin \text{Equal}_6$ and we will later argue that the error probability ε (i.e., the probability of obtaining $\ell = 0$) is at most $1/3$. This clearly places Equal_6 in BPCFL.

Let us assume that $w \notin \text{Equal}_6$ and $\ell = 0$. For any two pairs $(x_1, y_1), (x_2, y_2) \in [N] \times [N]$ that force ℓ to be zero, we obtain $(\alpha_1 - \alpha_4) + x_i(\alpha_2 - \alpha_5) + y_i(\alpha_3 - \alpha_6) = 0$ for any index $i \in \{1, 2\}$. From these two equations, it follows that (*) $(x_1 - x_2)(\alpha_2 - \alpha_5) = (y_2 - y_1)(\alpha_3 - \alpha_6)$.

(1) Consider the case where $\alpha_2 = \alpha_5$ but $\alpha_3 \neq \alpha_6$. By (*), we conclude that $y_1 = y_2$; that is, there is a unique solution y for the equation $\ell = 0$. Hence, the total number of (x, y) that forces $\ell = 0$ is exactly N , and thus ε equals $1/N$, which is smaller than $1/3$. The case where $\alpha_2 \neq \alpha_5$ and $\alpha_3 = \alpha_6$ is similar.

(2) Consider the case where $\alpha_2 \neq \alpha_5$ and $\alpha_3 \neq \alpha_6$. There are two cases to consider separately.

(a) If $\alpha_2 - \alpha_5$ and $\alpha_3 - \alpha_6$ are relatively prime, then we conclude that $x = x'$ and $y = y'$ from (*). This indicates that there is a unique solution pair (x, y) for the equation $\ell = 0$. Thus, the error probability ε is $1/N^2$, implying $\varepsilon < 1/3$.

(b) If $\alpha_2 - \alpha_5 = \beta(\alpha_3 - \alpha_6)$ holds for a certain non-zero integer β , then it follows that $x - x' = \beta(y' - y)$. Since $|x - x'|, |y' - y| \leq 4$, there are at most 12 cases for (x, y, x', y') with $x > x'$ that satisfy $x - x' = \beta(y' - y)$ for a certain non-zero integer β . Hence, ε is at most $12/25$, which is obviously smaller than $1/3$.

(3) The case where $\alpha_1 \neq \alpha_4$, $\alpha_2 = \alpha_5$, and $\alpha_3 = \alpha_6$ never occurs because ℓ is always non-zero. \square

It is not clear whether BPCFL is located inside the CFL hierarchy, because a standard argument used to prove that $\text{BPP} \subseteq \Sigma_2^P \cap \Pi_2^P$ requires an amplification property but a ppda cannot, in general, amplify its success probability [8].

Since a relationship between BPCFL and $\Sigma_2^{\text{CFL}} \cap \Pi_2^{\text{CFL}}$ is not clear, we may resort to an oracle separation between those language families. For this purpose, we will introduce a *relativization* of the target language families. First, we introduce a relativization of BPCFL. Similar to CFL_m^A , BPCFL_m^A is defined simply by providing underlying ppda's with extra query tapes. Second, we define a relativized CFL hierarchy. Relative to this A , a *relativized CFL hierarchy* $\{\Delta_k^{\text{CFL},A}, \Sigma_k^{\text{CFL},A}, \Pi_k^{\text{CFL},A} \mid k \in \mathbb{N}\}$ consists of the following families: $\Delta_0^{\text{CFL},A} = \Delta_1^{\text{CFL},A} = \Sigma_0^{\text{CFL},A} = \Pi_0^{\text{CFL},A} = \text{DCFL}_T^A$, $\Sigma_1^{\text{CFL},A} = \text{CFL}_T^A$, $\Pi_k^{\text{CFL},A} = \text{co-}\Sigma_{m,k}^{\text{CFL},A}$, $\Delta_{k+1}^{\text{CFL},A} = \text{DCFL}_T(\Sigma_k^{\text{CFL},A})$, and $\Sigma_{k+1}^{\text{CFL},A} = \text{CFL}_T(\Pi_k^{\text{CFL},A})$ for all indices $k \geq 1$. As this definition hints, any oracle-dependent language L^A in $\Sigma_k^{\text{CFL},A}$ can be recognized by a chain of k T -reduction machines whose last machine makes queries directly to A . As for later reference, we refer to this chain of k machines as a *defining machine set* for L^A .

Proposition 4.17 *There exists a recursive oracle A such that $\text{BPCFL}_m^A \not\subseteq \Sigma_2^{\text{CFL},A}$.*

For the proof of Proposition 4.17, we need to consider certain non-uniform families of levelable Boolean circuits, where a circuit is *levelable* if (i) all nodes are partitioned into levels, (ii) edges exist only between adjacent levels, (iii) the first level of a circuit has the output node, (iv) the gates at the same level are of the same type, (v) two gates at two adjacent levels are of the same type, and (vi) all input nodes (labeled by literals) are at the same level (see, e.g., [6]). The notation $\text{CIR}_k(n, m)$ denotes the collection of all levelable circuits C that satisfy the following conditions: (1) C has *depth* k (i.e., k levels of gates), (2) the *top gate* (i.e., the root node) of C is *OR*, (3) C has alternating levels of *OR* and *AND*, (4) the *bottom fan-in* (i.e., the maximum fan-in of any bottom gate) of C is at most m , (5) the fan-in of any gate except for the bottom gates is at most n , and (6) there are at most n input variables.

Lemma 4.18 *Let L^A be any oracle-dependent language over alphabet Σ in $\Sigma_k^{\text{CFL},A}$, where A is any oracle over alphabet Θ . Let (M_1, M_2, \dots, M_k) be a defining machine set for L^A . Let a and c be two positive constants such that the running time of each M_i is bounded from above by $c|x|$ and any query word y produced by M_k*

has length at most $a|x|$, where x is an input string to L^A . For every length $n \in \mathbb{N}^+$ and every input $x \in \Sigma^n$, there exist a Boolean circuit C in $CIR_{k+1}(2^{an}, ck)$ such that (1) all variables $x_1, x_2, \dots, x_{\ell(n)}$ of C are strings (and their negations) included in $\Theta^{\leq an}$ and (2) for any oracle A , it holds that x is in L^A if and only if C outputs 1 on inputs $(\chi^A(x_1), \chi^A(x_2), \dots, \chi^A(x_{\ell(n)}))$.

Proof. We will prove this lemma by induction on $k \geq 1$. We begin with the base case $k = 1$. Let L^A be any oracle-dependent language in CFL_T^A , where A is any oracle. Let M, a, c satisfy the premise of the lemma. Fix n and $x \in \Sigma^n$. By an argument similar to the proof of Proposition 3.13, we can modify M so that, before starting writing the i th query word y_i , it must guess its oracle answer b_i and produces $b_i y_i \sharp$ on a query tape and, instead of making an actual query, it assumes that A returns b_i . After this modification, a string produced on a query tape must be of the form $b_1 y_1 \sharp b_2 y_2 \sharp \dots \sharp b_\ell y_\ell \sharp$. Let V_x be composed of all such query strings produced along accepting computation paths. Note that $\|V_x\| \leq 2^{2an}$ since M halts within time an .

Next, we will define a circuit C , which is an OR of ANDs, as follows. The top OR gate has edges labeled by strings in V_x . For each $y \in V_x$, an associated subcircuit D_y , consisting of an AND gate, has input nodes labeled by literals of the form $y_1^{(b_1)}, y_2^{(b_2)}, \dots, y_\ell^{(b_\ell)}$, where $y_i^1 = y_i$ and $y_i^{(1)} = \overline{y_i}$. Let $x_1, x_2, \dots, x_{m(n)}$ be all distinct variables (of the positive form) appearing in C . It is not difficult to verify that, for any oracle A , $x \in L^A$ if and only if $C(\chi^A(x_1), \chi^A(x_2), \dots, \chi^A(x_{m(n)})) = 1$.

Let us consider the induction step $k \geq 2$. Assume that L^A is in $CFL_T(B^A)$ for a certain oracle $B^A \in \Pi_{k-1}^{CFL, A}$. Let M be a T -reduction machine reducing L^A to B^A . By a similar argument as in the base case, we can modify so that it generates query words of the form $b_1 y_1 \sharp \dots \sharp b_\ell y_\ell \sharp$ without making actual queries. We set V_x to be the collection of all such strings produced along accepting computation paths. Since $\overline{B^A} \in \Sigma_{k-1}^{CFL, A}$, by the induction hypothesis, for each string $y \in V_x$, there exists a circuit D_y satisfying the lemma. Instead of D_y , we consider its dual circuit $\overline{D_y}$. Here, we define C to be an OR of all $\overline{D_y}$'s for any $y \in V_x$. A similar reasoning as in the base case shows that, for any oracle A , x is in L^A if and only if $C(\chi^A(x_1), \chi^A(x_2), \dots, \chi^A(x_{m(n)})) = 1$. \square

Our example language is $L^A = \{0^n \mid \|A \cap \Sigma^n\| > 2^{n-1}\}$, where $\Sigma = \{0, 1\}$. To guarantee that $L^A \in \text{BPCFL}_m^A$, we will aim at constructing an oracle A satisfying that either $\|A \cap \Sigma^n\| \leq 2^n/3$ or $\|\overline{A} \cap \Sigma^n\| \leq 2^n/3$ for every length $n \in \mathbb{N}^+$. This is done by recursively choosing a pair of reduction machines that define each language B^A in $\Sigma_2^{CFL, A}$ and by defining a large enough length $n \in \mathbb{N}$ and a set $A_n (= A \cap \Sigma^n)$ such that $0^n \in L^A \not\leftrightarrow 0^n \in B^A$.

Proof of Proposition 4.17. Let $\Sigma = \{0, 1\}$ and consider an example language $L^A = \{0^n \mid \|A \cap \Sigma^n\| > 2^{n-1}\}$. To guarantee that $L^A \in \text{BPCFL}_m^A$, we consider only oracles A that satisfy the condition (*) that either $\|A \cap \Sigma^n\| \leq 2^n/3$ or $\|\overline{A} \cap \Sigma^n\| \leq 2^n/3$ for every length $n \in \mathbb{N}^+$.

Next, we will construct an appropriate oracle A satisfying that $L^A \notin \Sigma_2^{CFL, A}$. For this purpose, we use Lemma 4.18. First, we enumerate all oracle-dependent languages in Σ_k^{CFL} and consider their corresponding depth-3 Boolean circuit families that satisfy all the conditions stated in Lemma 4.18.

Recursively, we choose such a circuit family and define a large enough length n and a set $A_n (= A \cap \Sigma^n)$. Initially, we set $n_0 = 0$ and $A_0 = \emptyset$. Assume that, at stage $i - 1$, we have already defined n_{i-1} and A_{i-1} . Let us consider stage i . Take the i th circuit family $\{C_n\}_{n \in \mathbb{N}}$ and two constants $a, c > 0$ given by Lemma 4.18. First, we set $n_i = \max\{n_{i-1} + 1, 2^{a'n_{i-1}} + 1, c' + 1\}$, where a' and c' are constants taken at stage $i - 1$. The choice of n_i guarantees that A_{n_i} is not affected by the behaviors of the circuits considered at stage $i - 1$.

In the rest of the proof, we will examine two cases.

(1) Consider the base case where the bottom fan-in is exactly 1. For each label $y \in \{0, 1\}^{an}$, let $Q(y)$ be the set of all input variables that appear in subcircuits connected to the top OR gate by a wire labeled y . In particular, $Q^+(y)$ (resp., $Q^-(y)$) consists of variables in $Q(y)$ that appear in positive form (resp., negative form). Let us consider two cases.

(a) Assume that there exists a string y_0 such that $\|Q^+(y_0)\| \leq 2^n/3$. In this case, we set A_n to be $Q^+(y_0)$. It is obvious that $0^n \notin L^A$ and $C_n(\chi^A(x_1), \dots, \chi^A(x_{2an})) = 1$.

(b) Assume that, for all y , $\|Q^+(y)\| > 2^{an}/3$. Recursively, we will choose at most $an/\log(3/2) + 1$ strings. At the first step, let $B_0 = \{0, 1\}^{an}$. Assume that B_{i-1} has been defined. We will define B_i as follows. Choose lexicographically the smallest string w such that the set $\{y \in B_{i-1} \mid w \in Q^+(y)\}$ has the largest cardinality. Finally, we define w_i be this string w and set $B_i = \{y \in B_{i-1} \mid w_i \notin Q^+(y)\}$. In what follows, we will show that $\|B_i\| \leq (2/3) \|B_{i-1}\|$.

Claim 12 $\|B_i\| \leq (2/3) \|B_{i-1}\|$.

Proof. Let d satisfy $\|\overline{B_i}\| = d \cdot \|B_{i-1}\|$. For each index $i \in [2^{an}]$, let $X_i = \{y \mid x_i \in Q^+(y)\}$. Since $\|Q^+(y)\| > 2^{an}/3$ for all y 's, it holds that $\sum_{i=1}^{2^{an}} \|X_i\| \cdot d \geq (2^{an}/3) \|B_{i-1}\|$. Note that $\sum_{i=1}^{2^{an}} \|X_i\| = 2^{an}$. Thus, we obtain $d \geq 1/3$. Since $\|B_i\| = \|B_{i-1}\| - \|\overline{B_i}\|$, it follows that $\|B_i\| \leq (2/3) \|B_{i-1}\|$. \square

From the above claim, it follows that $\|B_i\| \leq (2/3)^i \|B_0\| = (2/3)^i 2^{an}$. Let i_0 denote the minimal number such that $\|B_i\| = 0$. Since $i > an/\log(3/2)$ implies $\|B_i\| < 1$, we conclude that $i_0 \leq an/\log(3/2) + 1$. Now, we write W for the collection of all w_i 's ($1 \leq i \leq i_0$) defined in the above procedure. The desired A_n is defined to be $(\Sigma^{an} - W) \cup (\bigcup_y Q^-(y))$.

(2) Second, we will consider the case where the bottom fan-in is more than 1. To handle this case, we will use a special form of the so-called *switching lemma* to reduce this case to the base case.

A *restriction* is a map ρ from a set of n Boolean variables to $\{0, 1, *\}$. We define $\mathcal{R}_n^{\ell, q}$ to be the collection of *restrictions* ρ on a domain of n variables that have exactly ℓ unset variables and a q -fraction of the variables are set to be 1. For any circuit C , $bf(C)$ denotes the bottom fan-in of C .

Claim 13 [1] *Let C be a circuit of OR of ANDs with bottom fan-in at most r . Let $n > 0$, $s \geq 0$, $\ell = pn$, and $p \leq 1/7$. It holds that $\|\{\rho \in \mathcal{R}_n^{\ell, q} \mid \exists D : \text{AND of ORs}[bf(D) \geq s]\}\| < (2pr/q^2)^s \|\mathcal{R}_n^{\ell, q}\|$.*

Consider any subcircuit D , an AND of ORs, attached to the top OR-gate. By setting $q = 1/3$ and $r = c$, we apply Claim 13 to D . The probability that D is written as an OR of ANDs with bottom fan-in at most an is upper-bounded by $1 - (18pc)^{an}$. Moreover, the probability that all such subcircuits D are simultaneously written as circuits, each of which is an AND of ORs, is at most $[1 - (18pc)^{an}]^{2^{an}} \geq 1 - 2^{an}(18pc)^{an} = 1 - (36pc)^{an}$. If we choose $p = 1/72c$, then the success probability is at least $1 - (36pc)^{an} \geq 1 - (1/2)^{an}$, which is larger than $1/2$ for any integer $n \geq 2/a$. Since every subcircuit D is written as an AND of ORs, the original circuit C can be written as an OR of ANDs with bottom fan-in at most an . Finally, we apply the base case to this new circuit. \square

There also exists an obvious oracle for which BPCFL equals Σ_2^{CFL} since the following equivalence holds.

Proposition 4.19 $\text{BPCFL}_T^{\text{PSPACE}} = \Sigma_2^{\text{CFL, PSPACE}} = \text{PSPACE}$.

Proof. It is obvious that $\text{BPCFL}_T^B \subseteq \text{PSPACE}_T^B$ for every oracle B , where PSPACE_T^B is a many-one relativization of PSPACE relative to B . Hence, it follows that $\text{BPCFL}_T^{\text{PSPACE}} \subseteq \text{PSPACE}$. Conversely, it holds that $\text{PSPACE} \subseteq \text{BPCFL}_T^{\text{PSPACE}}$. The case of $\Sigma_2^{\text{CFL, PSPACE}}$ is similar. \square

We have just seen an oracle that supports the inclusion $\text{BPCFL} \subseteq \Sigma_2^{\text{CFL}}$ and another oracle that does not. As this example showcases, some relativization results are quite counterintuitive. Before closing this subsection, we will present another plausible example regarding a *parity NFA language family* $\oplus\text{NFA}$ whose elements are languages of the form $\{x \mid \|ACC_M(x)\| \equiv 1 \pmod{2}\}$ for arbitrary nfa's M . In the unrelativized world, it is known that $\oplus\text{NFA} \subseteq \text{TC}^1 \subseteq \text{PH}$; however, there exists another oracle that defies this fact.

Lemma 4.20 *There exists an oracle such that $\oplus\text{NFA}_m^A \not\subseteq \text{PH}^A$.*

Proof. Let us consider a special language $L^A = \{0^n \mid \bigoplus_{x \in \Sigma^n} \chi^A(x) \equiv 1 \pmod{2}\}$ relative to oracle A . It is easy to show that, for any oracle A , L^A is in $\oplus\text{NFA}_m^A$ by guessing a string x in Σ^n and querying it to A . Since it is shown in [5] that $L^A \notin \text{PH}^A$ for a *random oracle* A , we immediately obtain the desired oracle separation. \square

5 A Close Relation to the Polynomial Hierarchy

Through the last section, the CFL hierarchy has proven to be viable in classifying certain languages and it can be characterized by two natural ways, as shown in Theorem 4.11. Moreover, we known that the first two levels of the CFL hierarchy are different (namely, $\Sigma_1^{\text{CFL}} \neq \Sigma_2^{\text{CFL}}$); however, the separation of the rest of the hierarchy still remains unknown. In this section, we will discuss under what conditions the separation is possible.

Given a language A , a language L is in L_m^A if there exists a logarithmic-space (or log-space) DTM M with an extra write-only query tape (other than a two-way read-only input tape and a two-way read/write work

tape) such that, for every string x , x is in L if and only if M on the input x produces a string in A . Recall that any tape head on a write-only tape moves only in one direction. For a language family \mathcal{C} , we denote by $L_m^{\mathcal{C}}$ the union $\bigcup_{A \in \mathcal{C}} L_m^A$. Occasionally, we also write $L_m(\mathcal{C})$ to mean $L_m^{\mathcal{C}}$. In particular, when $\mathcal{C} = \text{CFL}$, the language family L_m^{CFL} has been known as LOGCFL (LogCFL or LOG(CFL)) in the literature. Since $\Sigma_1^{\text{CFL}} = \text{CFL}$, it follows from [18] that $L_m(\Sigma_1^{\text{CFL}}) = \text{SAC}^1$. As for the language family PCFL, for instance, Macarie and Ogihara [13] demonstrated that $L_m^{\text{PCFL}} \subseteq \text{TC}^1$. Concerning BPCFL, the containment $L_m^{\text{BPCFL}} \subseteq L_m(\Sigma_2^{\text{CFL}})$ holds. This is shown as follows. From $\text{BPCFL} \subseteq \text{PCFL}$, it follows that $L_m^{\text{BPCFL}} \subseteq L_m^{\text{PCFL}} \subseteq \text{TC}^1$. However, by Claim 14, $L_m(\Sigma_2^{\text{CFL}}) = \text{NP}$. Thus, it holds that $L_m^{\text{BPCFL}} \subseteq \text{TC}^1 \subseteq \text{NP} = L_m(\Sigma_2^{\text{CFL}})$.

It is obvious that $\mathcal{C}_1 = \mathcal{C}_2$ implies $L_m^{\mathcal{C}_1} = L_m^{\mathcal{C}_2}$; however, the converse does not always hold. Here is a simple example. Although $\text{CFL}(k) \neq \text{CFL}$ holds for $k \geq 2$, the following equality holds.

Lemma 5.1 $L_m^{\text{CFL}(\omega)} = L_m^{\text{CFL}} = \text{SAC}^1$.

Proof. As noted before, it is known that $L_m^{\text{CFL}} = \text{SAC}^1$. Therefore, our goal is set to prove that $L_m^{\text{CFL}(k)} = L_m^{\text{CFL}}$ for any index $k \geq 2$. Note that $L_m^{\text{CFL}} = L_m^{\text{CFL}(1)} \subseteq L_m^{\text{CFL}(k)}$ for all indices $k \in \mathbb{N}^+$. The remaining task is to show that $L_m^{\text{CFL}(k)} \subseteq L_m^{\text{CFL}}$. Let $k \geq 2$ and assume that $A \in L_m^B$ for a certain language $B \in \text{CFL}(k)$. There are k languages $B_1, B_2, \dots, B_k \in \text{CFL}$ satisfying that $B = \bigcap_{i \in [k]} B_i$. Take any log-space oracle DTM M that recognizes A relative to B .

We define a new reduction machine N_1 so that, on input x , it outputs $y_1 \# y_2 \# \dots \# y_k$ (k y 's) if M on input x outputs y . Define $C = \{y_1 \# y_2 \# \dots \# y_k \mid \forall i \in [k] [y_i \in B_i]\}$. Obviously, it holds that $N_1(x) \in C$ iff $M(x) \in B$. It thus suffices to show that C is in CFL.

To show that $C \in \text{CFL}$, let us consider an npda N_2 that behaves as follows. On input $w = y_1 \# y_2 \# \dots \# y_k$, N_2 sequentially simulates M_i on input y_i , starting with $i = 1$. After each simulation of $M_i(y_i)$, we always clear the stack so that each simulation does not affect the next one. Moreover, as soon as M_i rejects y_i , N_2 enters a rejecting state. It is obvious that C is recognized by N_2 . \square

The CFL hierarchy turns out to be a useful tool because it is closely related to the polynomial hierarchy $\{\Delta_k^P, \Sigma_k^P, \Pi_k^P \mid k \in \mathbb{N}\}$. Reinhardt [14] first established a close connection between his alternating hierarchy over CFL and the polynomial hierarchy. Similar to $\Sigma_{k,e}^{\text{CFL}}$, the notation $\Sigma_{k,e}^P$ stands for the e th level of the Boolean hierarchies over Σ_k^P . We want to demonstrate the following intimate relationship between $\Sigma_{k+1,e}^{\text{CFL}}$ and $\Sigma_{k,e}^P$.

Theorem 5.2 For every index $e, k \in \mathbb{N}^+$, $L_m(\Sigma_{k+1,e}^{\text{CFL}}) = \Sigma_{k,e}^P$ holds.

Proof. Fixing k arbitrarily, we will show the theorem by induction on $e \in \mathbb{N}^+$. Our starting point is the base case where $e = 1$.

Claim 14 $L_m(\Sigma_{k+1}^{\text{CFL}}) = \Sigma_k^P$ holds for every index $k \in \mathbb{N}^+$.

Proof. We want to demonstrate separately that, for every index $k \in \mathbb{N}^+$, (1) $L_m(\Sigma_{k+1}^{\text{CFL}}) \subseteq \Sigma_k^P$ and (2) $\Sigma_k^P \subseteq L_m(\Sigma_{k+1}^{\text{CFL}})$.

(1) To prove that $L_m(\Sigma_{k+1}^{\text{CFL}}) \subseteq \Sigma_k^P$, we start with the following useful relationship between $\Sigma_{k+1}^{\text{CFL}}$ and Σ_k^P .

Claim 15 $\Sigma_{k+1}^{\text{CFL}} \subseteq \Sigma_k^P$ holds for every index $k \in \mathbb{N}^+$.

Proof. This claim will be proven by induction on $k \geq 1$. A key to the following proof is the fact that $\text{CFL}_T^A \subseteq \text{NP}^A$ holds for every oracle A . When $k = 1$, it holds that $\Sigma_2^{\text{CFL}} = \text{CFL}_T^{\text{CFL}} \subseteq \text{NP}^{\text{CFL}}$. Since $\text{CFL} \subseteq \text{P}$, we obtain $\text{NP}^{\text{CFL}} \subseteq \text{NP}^P = \text{NP}$, yielding the desired containment $\Sigma_2^{\text{CFL}} \subseteq \text{NP}$. When $k \geq 2$, by the induction hypothesis, we assume that $\Sigma_k^{\text{CFL}} \subseteq \Sigma_{k-1}^P$. It therefore follows that $\Sigma_{k+1}^{\text{CFL}} = \text{CFL}_T(\Pi_k^{\text{CFL}}) \subseteq \text{NP}(\Pi_k^{\text{CFL}}) \subseteq \text{NP}(\Pi_{k-1}^P) = \Sigma_k^P$. \square

The inclusion $L_m(\Sigma_{k+1}^{\text{CFL}}) \subseteq L_m(\Sigma_k^P)$ follows from Claim 15. Hence, using the fact that $L_m(\Sigma_k^P) \subseteq \Sigma_k^P$, we conclude that $L_m(\Sigma_{k+1}^{\text{CFL}}) \subseteq \Sigma_k^P$.

(2) Next, we will show that $\Sigma_k^P \subseteq L_m(\Sigma_{k+1}^{\text{CFL}})$. An underlying idea of the following argument comes from [14]. Our plan is to define a set of k quantified Boolean formulas, denoted QBF_k , which is slightly different from a standard BQF_k , and to prove that (a) QBF_k is log-space complete for Σ_k^P and (b) QBF_k indeed belongs to $\Sigma_{k+1}^{\text{CFL}}$. Combining (a) and (b) implies that $\Sigma_k^P \subseteq L_m^{QBF_k} \subseteq L_m(\Sigma_{k+1}^{\text{CFL}})$.

First, we will discuss the case where k is odd. The language QBF_k must be of a specific form so that an input-tape head of an oracle npda can read through a given instance of QBF_k from left to right without back-tracking. First, we prepare the following alphabet of distinct input symbols: $\Sigma_k = \{\exists, \forall, \wedge, \neg, +, -, 0, a_1, a_2, \dots, a_k\}$. A string ϕ belongs to QBF_k exactly when ϕ is of the following form: $\exists a_1^{m_1} \forall a_2^{m_2} \dots Q_k a_k^{m_k} \neg c_1 \wedge c_2 \wedge \dots \wedge c_m$, where each m_i and m are in \mathbb{N}^+ , Q_k is \exists , each c_i is a string $c_{i,\ell_i} c_{i,\ell_i-1} \dots c_{i,1}$ in $\{+, -, 0\}^{\ell_i}$ for a certain number ℓ_i satisfying $\ell_i \geq \overline{m} = \sum_{j=1}^k m_j$, and, moreover, the corresponding quantified Boolean formula

$$\tilde{\phi} \equiv \exists x_1, \dots, x_{m_1} \forall x_{m_1+1}, \dots, x_{m_1+m_2} \dots Q_k x_{m'+1}, \dots, x_{m'+m_k} [C_1 \wedge C_2 \wedge \dots \wedge C_m]$$

is satisfied, where $m' = \overline{m} - m_k$, C_i is a formula of the form $(\bigvee_{j \in S_+(i)} x_j) \vee (\bigvee_{j \in S_-(i)} \overline{x_j})$, $S_+(i) = \{j \in [\ell] \mid j \leq \overline{m}, c_{i,\ell_i-j+1} = +\}$, and $S_-(i) = \{j \in [\ell] \mid j \leq \overline{m}, c_{i,\ell_i-j+1} = -\}$.

When k is even, we define BQF_k by exchanging the roles of \wedge and \vee and by setting $Q_k = \forall$ in the above definition. As is shown in [15], it is possible to demonstrate that QBF_k is log-space many-one complete for Σ_k^P ; that is, every language in Σ_k^P belongs to $L_m^{QBF_k}$. If QBF_k is in Σ_{k+1}^{CFL} , then we obtain $\Sigma_k^P \subseteq L_m^{QBF_k} \subseteq L_m(\Sigma_{k+1}^{CFL})$, as requested.

Therefore, what remains undone is to prove that QBF_k is indeed in Σ_{k+1}^{CFL} by constructing a chain of m -reduction machines for QBF_k . As done in Section 4.3, we also call such a chain of m -reduction machines computing QBF_k by a *defining machine set* for QBF_k . Given an index $i \in [k]$, we define a string ϕ_i as $\phi_i \equiv Q_i a_i^{m_i} \dots Q_k a_k^{m_k} \neg c_1 \wedge c_2 \wedge \dots \wedge c_m$. Now, let ψ_1 denote ϕ_1 . The $(i+1)$ st machine works as follows. Assume that an input string ψ_i to the machine has the form $[a_{s_1}^{m_1}] \dots [a_{s_{i-1}}^{m_{i-1}}] \phi_i$. While reading $m_i + 1$ symbols of " $Q_i a_i^{m_i}$ " until the next symbol Q_{i+1} , the machine generates all strings $s_i = s_{i1} s_{i2} \dots s_{im_i}$ in $\{0, 1\}^{m_i}$ and then produces corresponding strings $[a_{s_1}^{m_1}] \dots [a_{s_i}^{m_i}] \phi_{i+1}$ on a query tape. Note that, at any moment, if the machine discovers that the input does not have a valid form, it immediately enters a rejecting state and halts. The last machine N works in the following manner, when ψ_k is given as an input. First, N stores the string $[a_{s_1}^{m_1}] \dots [a_{s_{k-1}}^{m_{k-1}}]$ in its stack, guesses a binary string s_k of length m_k , and stores $[a_{s_k}^{m_k}]$ also in the stack. Second, N guesses j , locates the block of c_j , and checks whether its corresponding Boolean formula C_j is satisfied by the assignment specified by $s_1 s_2 \dots s_k$. This checking process can be easily done by comparing the two strings $s_1 s_2 \dots s_k$ and c_j symbol by symbol as follows: if s_{ij} corresponds to c_{i,ℓ_i-j+1} , then N accepts the input exactly when $(s_{ij} = 1 \wedge c_{i,\ell_i-j+1} = +)$ or $(s_{ij} = 0 \wedge c_{i,\ell_i-j+1} = -)$. The existence of a defining machine set for QBF_k proves that QBF_k indeed belongs to Σ_{k+1}^{CFL} . \square

Finally, let us consider the case where $e \geq 2$. A key to the proof of this case is the following simple fact. For convenience, we say that a language family \mathcal{C} *admits input redundancy* if, for every language L in \mathcal{C} , two languages $L' = \{x \neg y \mid x \in L\}$ and $L'' = \{x \neg y \mid y \in L\}$ are both in \mathcal{C} , provided that \neg is a fresh symbol that never appears in x as well as y .

Claim 16 *If two language families \mathcal{C}_1 and \mathcal{C}_2 admit input redundancy, then $L_m^{C_1} \wedge L_m^{C_2} \subseteq L_m^{C_1 \wedge C_2}$ and $L_m^{C_1} \vee L_m^{C_2} \subseteq L_m^{C_1 \vee C_2}$.*

Proof. We will show only the first assertion of the lemma, because the second assertion follows similarly. Take any language L and assume that $L \in L_m^{A_1} \wedge L_m^{A_2}$ for certain two languages $A_1 \in \mathcal{C}_1$ and $A_2 \in \mathcal{C}_2$. there are two log-space m -reduction machines M_1 and M_2 such that, for every string x , x is in L if and only if $M_i^A(x)$ outputs y_i and $y_i \in A_i$ for any index $i \in \{1, 2\}$. Now, we want to define another machine M as follows. On input x , M first simulates M_1 on x and produces $y_1 \neg$ on its query tape. Moreover, M simulates M_2 on x and produces y_2 also on the query tape following the string $y_1 \neg$. The language $C = \{y_1 \neg y_2 \mid y_1 \in A_1, y_2 \in A_2\}$ clearly belongs to $\mathcal{C}_1 \wedge \mathcal{C}_2$. Thus, it is obviously that L is in $L_m^C \subseteq L_m^{C_1 \wedge C_2}$. \square

Claim 16 implies that $L_m(\Sigma_{k+1,2e+1}^{CFL}) = L_m(\Sigma_{k+1,2e}^{CFL} \vee \Sigma_{k+1}^{CFL}) = L_m(\Sigma_{k+1,2e}^{CFL}) \vee L_m(\Sigma_{k+1}^{CFL})$. Since $L_m(\Sigma_{k+1}^{CFL}) = \Sigma_k^P$ and $L_m(\Sigma_{k+1,2e}^{CFL}) = \Sigma_{k,2e}^P$ by the induction hypothesis, we obtain $L_m(\Sigma_{k+1,2e+1}^{CFL}) = \Sigma_{k,2e}^P \vee \Sigma_k^P = \Sigma_{k,2e+1}^P$. Similarly, we conclude that $L_m(\Sigma_{k+1,2e+2}^{CFL}) = \Sigma_{k,2e+2}^P$. \square

Unfortunately, the proof presented above does not apply to obtain, for instance, $L_m(\Sigma_{k+1}^{CFL} \cap \Pi_{k+1}^{CFL}) = \Sigma_k^P \cap \Pi_k^P$, simply because no many-one complete languages are known for $\Sigma_k^P \cap \Pi_k^P$. This remains as a challenging question.

An immediate consequence of Theorem 5.2 is given below.

Corollary 5.3 *If the polynomial hierarchy is infinite, then so is the Boolean hierarchy over Σ_k^{CFL} at every level $k \geq 2$. In particular, if the polynomial hierarchy is infinite, then so is the CFL hierarchy.*

Proof. Let $k \geq 2$. Theorem 5.2 implies that, if the Boolean hierarchy over Σ_k^{P} is infinite, then the Boolean hierarchy over $\Sigma_{k+1}^{\text{CFL}}$ is also infinite. Earlier, Kadin [9] showed that, if the polynomial hierarchy is infinite, then the Boolean hierarchy over Σ_k^{P} is infinite for every index $k \geq 1$. By combining those statements, we instantly obtain the desired consequence. \square

More specifically, the following separation holds for the k th level and the $(k+1)$ st level of the CFL hierarchy.

Corollary 5.4 *Let $k \geq 2$. If $\text{PH} \neq \Sigma_k^{\text{P}}$, then $\Sigma_k^{\text{CFL}} \neq \Sigma_{k+1}^{\text{CFL}}$.*

In terms of the inclusion relationship, as shown in Claim 15, $\Sigma_{k+1}^{\text{CFL}}$ is upper-bounded by Σ_k^{P} . It is possible to show that this bound is tight under the assumption that Δ_{k+1}^{P} is different from Σ_{k+1}^{P} .

Corollary 5.5 *For any index $k \geq 2$, if $\Delta_{k+1}^{\text{P}} \neq \Sigma_{k+1}^{\text{P}}$, then $\Sigma_{k+2}^{\text{CFL}} \not\subseteq \Sigma_k^{\text{P}}$.*

Proof. We want to show the contrapositive of the corollary. We start with the assumption that $\Sigma_{k+2}^{\text{CFL}} \subseteq \Sigma_k^{\text{P}}$. From this inclusion, it follows that $L_m(\Sigma_{k+2}^{\text{CFL}}) \subseteq L_m(\Sigma_k^{\text{P}})$. Since $L_m(\Sigma_{k+2}^{\text{CFL}}) = \Sigma_{k+1}^{\text{P}}$ holds by Theorem 5.2, we obtain $\Sigma_{k+1}^{\text{P}} \subseteq L_m(\Sigma_k^{\text{P}})$. On the contrary, it holds that $L_m(\Sigma_k^{\text{P}}) \subseteq \Sigma_k^{\text{P}} = \Delta_{k+1}^{\text{P}}$. As a result, Σ_{k+1}^{P} is included in Δ_{k+1}^{P} . Since $\Delta_{k+1}^{\text{P}} \subseteq \Sigma_{k+1}^{\text{P}}$ is obvious, $\Delta_{k+1}^{\text{P}} = \Sigma_{k+1}^{\text{P}}$ follows immediately. \square

Let us recall that $\text{CFL}(\omega) \subseteq \Sigma_2^{\text{CFL}} \cap \Pi_2^{\text{CFL}}$ by Proposition 4.7 and $\text{CFL}(\omega) \subseteq \text{SAC}^1 \subseteq \text{NC}^2$ by Lemma 5.1. If $\Sigma_2^{\text{CFL}} \subseteq \text{NC}^2$ holds, then Claim 15 implies that $\text{NP} = L_m(\Sigma_2^{\text{CFL}}) \subseteq L_m(\text{NC}^2) = \text{NC}^2$, and we obtain $\text{NP} = \text{NC}^2$ since $\text{NC}^2 \subseteq \text{NP}$ is obvious. In the end, we reach the following conclusion.

Corollary 5.6 *If $\text{NP} \neq \text{NC}^2$, then $\Sigma_2^{\text{CFL}} \not\subseteq \text{NC}^2$.*

We have focused our attention to the computational complexity of languages that are log-space many-one reducible to certain languages in Σ_k^{CFL} . In the rest of this section, we will turn our attention to log-space truth-table reducible languages to Σ_k^{CFL} . Here, we use the notation L_{tt}^A (or $L_{tt}(A)$) to mean a family of languages that are log-space truth-table reducible to A . It is easy to show that a truth-table reduction can simulate Boolean operations that define each level of the Boolean hierarchy over CFL. Hence, the Boolean hierarchy BHCFL is “equivalent” to CFL under the log-space truth-table reducibility.

Lemma 5.7 $L_{tt}^{\text{BHCFL}} = L_{tt}^{\text{CFL}}$.

Proof. We need to show the equality $L_{tt}^{\text{CFL}_k} = L_{tt}^{\text{CFL}}$ for every index $k \geq 1$. Since $\text{CFL} \subseteq \text{CFL}_k$, it holds that $L_{tt}^{\text{CFL}} \subseteq L_{tt}^{\text{CFL}_k}$. Conversely, we will show that $L_{tt}^{\text{CFL}_k} \subseteq L_{tt}^{\text{CFL}}$ by induction on $k \geq 1$. Since the base case $k = 1$ is trivial, we hereafter assume that $k \geq 2$. Let L be any language in $L_{tt}^{\text{CFL}_k}$. Moreover, let M be a log-space oracle DTM M and A be an oracle A in CFL_k such that M tt -reduces L to A . Now, consider the case where k is even. We will construct a new oracle npda N as follows. On input x , when M produces m query words y_1, y_2, \dots, y_m , N produces $2m$ query words $0y_1, 1y_1, 0y_2, 1y_2, \dots, 0y_m, 1y_m$. Assume that $A = B \cap C$ for two appropriate languages $B \in \text{CFL}_{k-1}$ and $C \in \text{co-CFL}$. We define $B' = \{0y \mid y \in B\}$ and $C' = \{1y \mid y \notin C\}$ and we set A' to be $B' \cup C'$, which is in $\text{CFL}_{k-1} \vee \text{CFL} = \text{CFL}_{k-1}$ since CFL_{k-1} is closed under union with CFL. Note that y_i is in A if and only if $0y_i \notin B'$ and $1y_i \in C'$. We use this equivalence relation as a truth table to judge the membership of x to L .

When k is odd, since $A = B \cup C$ for certain languages $B \in \text{CFL}_{k-1}$ and $C \in \text{CFL}$, it suffices to transform B and C to $B' = \{0y \mid y \in B\}$ and $C' = \{1y \mid y \in C\}$. The rest of the argument is similar to the previous case. \square

Wagner [19] introduced a convenient notation Θ_{k+1}^{P} as an abbreviation of $P_T(\Sigma_k^{\text{P}}[O(\log n)])$ for each level $k \geq 1$, where the script “ $[O(\log n)]$ ” means that the total number of queries made in an entire *computation tree* on an input of size n to an oracle in Σ_k^{P} is bounded from above by $c \log n + d$ for two absolute constants $c, d \geq 0$.

Theorem 5.8 *For all levels $k \geq 1$, $L_{tt}(\Sigma_{k+1}^{\text{CFL}}) = \Theta_{k+1}^{\text{P}}$ holds.*

Proof. Let $k \geq 1$. First, we will give a useful characterization of Θ_{k+1}^{P} in terms of Σ_k^{P} using two

different truth-table reductions. In a way similar to L_{tt}^A , the notation P_{tt}^A (or $P_{tt}(A)$) is introduced using polynomial-time DTMs instead of log-space DTMs.

Claim 17 For every index $k \in \mathbb{N}^+$, it holds that $\Theta_{k+1}^P = P_{tt}(\Sigma_k^P) = L_{tt}(\Sigma_k^P)$.

Proof. It suffices to show that $P_{tt}(\Sigma_k^P) \subseteq L_{tt}(\Sigma_k^P)$ and $\Theta_{k+1}^P = P_{tt}(\Sigma_k^P)$ since $L_{tt}(\Sigma_k^P) \subseteq P_{tt}(\Sigma_k^P)$ is obvious. Note that the proof of $P_{tt}^{NP} \subseteq L_{tt}^{NP}$ by Buss and Hay [4] relativizes; namely, $P_{tt}(NP^A) \subseteq L_{tt}(NP^A)$ for any oracle A . Recall the language QBF_k defined in the proof of Claim 15. By choosing QBF_{k-1} for A , we obtain $P_{tt}(\Sigma_k^P) = P_{tt}(NP^{QBF_{k-1}}) \subseteq L_{tt}(NP^{QBF_{k-1}}) = L_{tt}(\Sigma_k^P)$. Moreover, the proof of $P_T^{NP[O(\log n)]} = P_{tt}^{NP}$ given in, e.g., [4] also relativizes. By a similar argument as above, it also follows that $P_T(\Sigma_k^P[O(\log n)]) = P_{tt}(\Sigma_k^P)$. \square

Since we have earlier shown that $\Sigma_{k+1}^{CFL} \subseteq \Sigma_k^P$, it follows that $L_{tt}(\Sigma_{k+1}^{CFL}) \subseteq L_{tt}(\Sigma_k^P) = \Theta_{k+1}^P$, where the last equality comes from Claim 17. In what follows, we intend to argue that $\Theta_{k+1}^P \subseteq L_{tt}(\Sigma_{k+1}^{CFL})$. Assume that $L \in \Theta_{k+1}^P$; thus, L is in $L_{tt}(\Sigma_k^P)$ by Claim 17. Since QBF_k is log-space many-one complete for Σ_k^P , we can replace Σ_k^P with QBF_k . Since $\Sigma_k^P \subseteq L_{tt}^{QBF_k}$, it holds that $L \in L_{tt}(L_{tt}^{QBF_k}) = L_{tt}^{QBF_k}$. Since QBF_k belongs to Σ_{k+1}^{CFL} by the proof of Claim 15, we conclude that $L \in L_{tt}(\Sigma_{k+1}^{CFL})$. \square

References

- [1] P. W. Beame. Lower bounds for recognizing small cliques on CRCW PRAM's. *Discrete Applied Mathematics*, 29 (1990) 3–20.
- [2] R. Book and K. Ko. On sets truth-table reducible to sparse sets. *SIAM J. Comput.* 17 (1988) 903–919.
- [3] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM J. Comput.* 13 (1984) 461–487.
- [4] S. R. Buss and L. Hay. On the truth-table reducibility to SAT. *Inf. Comput.* 91 (1991) 86–102.
- [5] J.-Y. Cai. With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy. In *Proc. of the 18th ACM Symposium on Theory of Computing*, pp.21–29, 1986.
- [6] D. Du and K. Ko. *Theory of Computational Complexity*, John Wiley & Sons, 2000.
- [7] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Second Edition. Addison-Wesley, 2001.
- [8] J. Hromkovič and G. Schnitger. On probabilistic pushdown automata. *Inf. Comput.* 208 (2010) 982–995.
- [9] J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM J. Comput.* 17 (1988) 1263–1282. Erratum appears in *SIAM J. Comput.* 20 (1991) p.404.
- [10] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial-time reducibilities. *Theor. Comput. Sci.* 1 (1975) 103–123.
- [11] P. Linz. *An Introduction to Formal Languages and Automata*. Fourth Edition. Jones and Barlett Publishers, 2006.
- [12] L. Y. Liu and P. Weiner. An infinite hierarchy of intersections of context-free languages. *Math. Systems Theory* 7 (1973) 185–192.
- [13] I. I. Macarie and M. Ogihara. Properties of probabilistic pushdown automata. *Theor. Comput. Sci.* 207 (1998) 117–130.
- [14] K. Reinhardt. Hierarchies over the context-free languages. In *Proc. of the 6th International Meeting of Young Computer Scientists on Aspects and Prospects of Theoretical Computer Science (IMYCS)*, Lecture Notes in Computer Science, Springer, vol.464, pp.214–224, 1990.
- [15] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.* 3 (1976) 1–22.
- [16] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. of the 5th Annual ACM Symposium on Theory of Computing*, pp.1–9, 1973.
- [17] K. Tadaki, T. Yamakami, and J. C. H. Lin. Theory of one-tape linear-time Turing machines. *Theor. Comput. Sci.* 411 (2010) 22–43. An extended abstract appeared in the Proc. of the 30th SOFSEM Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2004), Lecture Notes in Computer Science, Springer, vol.2932, pp.335–348, 2004.
- [18] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. System Sci.* 42 (1991) 380–404.
- [19] K. W. Wagner. Bounded query classes. *SIAM J. Comput.* 19 (1990) 833–846.
- [20] T. Yamakami. Swapping lemmas for regular and context-free languages. Available at arXiv:0808.4122, 2008.

- [21] T. Yamakami. Pseudorandom generators against advised context-free languages. Available at arXiv:0902.2774, 2009.
- [22] T. Yamakami. The roles of advice to one-tape linear-time Turing machines and finite automata. *Int. J. Found. Comput. Sci.* 21 (2010) 941–962. An early version appeared in the Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009), Lecture Notes in Computer Science, Springer, vol.5878, pp.933–942, 2009.
- [23] T. Yamakami. Immunity and pseudorandomness of context-free languages. *Theor. Comput. Sci.* 412 (2011) 6432–6450.
- [24] T. Yamakami and Y. Kato. The dissecting power of regular languages. *Inf. Pross. Lett.* 113 (2013) 116–122.
- [25] D. H. Young. Recognition and parsing of context-free languages in time n^3 . *Inf. Control* 10 (1967) 189–208.